École polytechnique de Louvain

Resilience of Tor relays on BGP hijacks & BGP monitoring of potential hijacks containing Tor prefixes

Author: **William VISÉE**
Supervisors: **Ramin SADRE, Florentin ROCHET**
Reader: **Olivier PEREIRA**
Academic year 2019–2020
Master [120] in Computer Science

Abstract

Tor is a network that provides anonymity and security to its users from untrusted destinations and third parties on the Internet. It is one of the largest anonymity networks deployed. Within the proposed metrics on the *Tor Metrics* website, none are about security. It would be valuable for the community if security-related metrics were proposed. In a correlation attack, if an attacker has access to traffic going in and out of the Tor network, he can correlate both traffics and the user loses his anonymity. To be able to see both sides of the traffic, the attacker can make BGP hijack to redirect traffic. Defenses exist for this type of attack, but none can definitively counter the attack at this time. This master thesis proposes new security metrics for the Tor network, in order to tell which part of the network is more robust against BGP hijack, giving hence a visualization of security threat to the users.

Acknowledgements

First, I would like to thanks my thesis supervisors assistant Florentin Rochet and Professor Ramin Sadre for their guidance, encouragement, and availability through the development of this master thesis. It was not always easy for me to work on this paper and they motivated me by giving me deadlines and goals on the work to accomplish.

Then, I would also like to thanks professor Olivier Bonaventure and assistant Florentin Rochet for guiding me towards a subject that I liked.

Finally, I would like to thanks Professor Olivier Pereira for accepting reading this document and be part of my jury.

Contents

Table of Acronyms

This table lists the main acronyms used in this document.

TOR	The Onion Routing
RAPTOR	Routing Attacks on Privacy in TOR
BGP	Border Gateway Protocol
AS	Autonomous System
ASN	Autonomous System Number
IP	Internet Protocol
TCP	Transmission Control Protocol
TLS	Transport Layer Security
ISP	Internet Service Provider
IANA	Internet Assigned Numbers Authority
RIR	Regional Internet Registry
ID	Identifier
MED	Multiple Exit Discriminator
DNS	Domain Name System
RPKI	Resource Public Key Infrastructure
ROA	Route Origin Authorization
API	Application Programming Interface
RIB	Routing Information Base
MRT	Multi-Threaded Routing Toolkit
HTML	HyperText Markup Language
URL	Uniform Resource Locator
TAR	Tape Archiver
OOM	Out Of Memory (killer)
RAM	Random Access Memory

CHAPTER 1

Introduction

The Onion Routing (Tor) is a network that provides anonymity and security to its users from untrusted destinations and third parties on the Internet [1]. It is one of the largest anonymity networks deployed [2]. Unfortunately, this service is not devoid of flaws. One big issue in Tor is correlation attacks. If an attacker has access to traffic going in and out of the Tor network, he can correlate both traffic and see if they are the same. If it is the case, the client loses his anonymity because the attacker has access to the source and destination of the communication.

Two families of defenses exist for this attack. The first one is about reducing the possibility of an attacker to observe both sides of traffic. The second is trying to make impossible correlation attacks even if you can see both sides of the traffic.

As Tor is oriented to provide a service with low latency, trying to make impossible correlation attacks by obfuscating packet timing/size or encrypt header will be too costly for the Tor properties. Consequently, we will look towards the solution of reducing the possibility of an attacker to observe ingress and egress traffic.

A set of attacks called RAPTOR [3] can benefit from the unsecured and dynamic properties of the Internet to make an adversary able to observe both sides of the traffic. RAPTOR is composed of an active attack called Border Gateway Protocol hijack (BGP hijack). A malicious Autonomous System (AS) will announce a prefix that it does not own to receive traffic intended for another AS. The malicious AS will be able to see the traffic and potentially do a correlation attack if it has got the other side of the traffic.

Multiple defenses exist but none can counter the attack definitively at the time [4]. This is why we want to propose security metrics for the Tor network to tell which part of the network is more robust against BGP hijack, giving hence a visualization of security threat to the users.

We will propose two security metrics from Tor creation to live time :

- The resilience of Tor relays on BGP hijacks. This means the proportion of ASes that will choose the path to the right origin when a malicious AS announces the Internet Protocol (IP) prefix containing the Tor relay. Each Tor relay will be given a resilience each first day of the month.
- A list of potential BGP hijack sorted chronologically. We will give the time of the announcement, the prefix announced, the potential malicious AS, and other information about how the announcement has been alerted.

The document starts by explaining Tor and BGP in chapter 2. Chapter 3 is composed of the state of the art. The motivation for the conception of the two programs will be described in chapter 4. The programs proposed to compute the metrics will be described in chapters 5 and 6. We will also analyze the result obtained in chapters 7. Chapter 8 will conclude this paper.

CHAPTER 2

Background

2.1 TOR: The Onion Routing

Tor is a service that allows its users to gain more privacy and security on the Internet. Tor protects you better from traffic analysis. Without Tor, a network adversary can analyze your traffic in order to know the two endpoints of a communication[5].

Usually, when you communicate with a server, the data exchanged pass through a Transmission Control Protocol (TCP) [6] link connecting the source and the destination. Anyone seeing this traffic can see the IP address of the client and the server by looking at the IP/TCP header. Even if the connection is encrypted, the header of the packets still remains clear. Eavesdroppers of this connection can now tell on which website/server a user/IP address is connected to.

For example, this problem can lead to (a) tracking on the web to make personalized advertisements, (b) discrimination based on your country not to be able to watch foreign data, (c) revealing who you are by analyzing the website on which you are going to, ... [5]

Tor network sits in the middle of your connection with a server. As a Tor client, you will choose three relays in which your connection will be passing through. There are called: guard relay, middle relay, and exit relay. As represented in figure 2.1, the Tor circuit will be as following: the Tor client -> Guard -> Middle -> Exit -> Server. The circuit works obviously both ways.

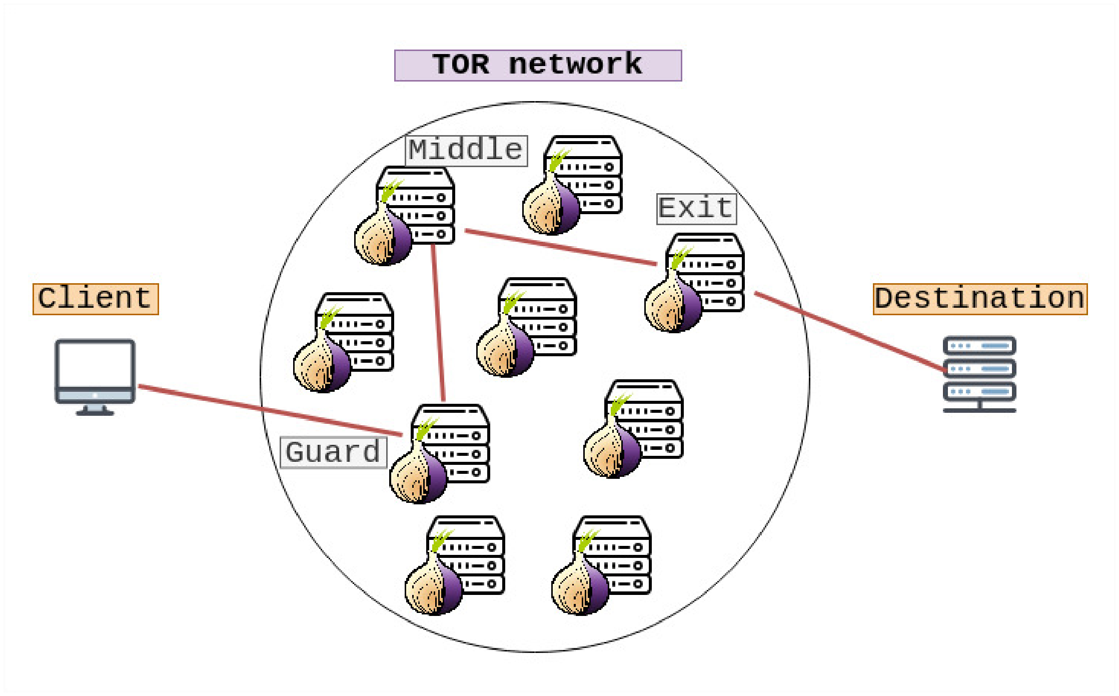


Figure 2.1: Illustration from a network point of view on a Tor client traffic path

Each relay in the circuit will only know the preceding and following nodes. This will ensure that none entity will know the IP address of the client and the IP address of the server. This is done by onion routing[7]. This technique of routing encapsulates the packets with multiple layers of encryption analogous to layers of an onion. The packet is sent over network nodes called onion routers, each of which "peels" away a single layer, uncovering the data's next destination. When the final layer is decrypted by the last router, the packet is sent to the destination.

In Tor, the client exchange symmetric secret keys with the three relays. When the client makes a request to a server, he will encapsulate the packets with the three secret keys in an order in which when a relay decrypts a layer of the packet, it will know the data to send to the next hop. Finally, the packets arrive to the server in clear (not encrypted by Tor). During the response, each relay will encapsulate back the packets from the server to the Tor client. Figure 2.2 shows how the packets are represented.

Remark: As Tor encapsulates the packet with encryption layers, the communication from the client to the exit relay is encrypted. The packets going from the exit relay to the destination are not encrypted by Tor because the packet has no more layer of encryption. However, if the client sets up a TLS communication with the destination, the egress traffic will be therefore encrypted.

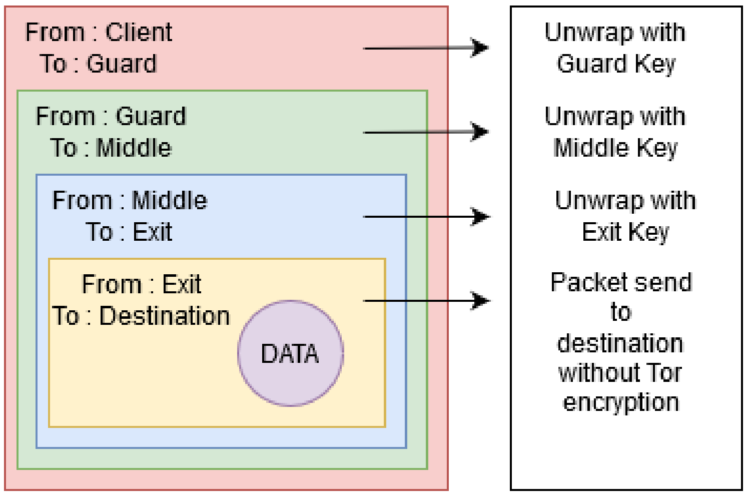


Figure 2.2: Example of a packet encapsulated with three layers of encryption. Each relay will decrypt a layer with their secret key and discover the next packet to be sent

Using an onion routing system like previously described, a network adversary will have more difficulties to make simple traffic analysis. Unfortunately, Tor cannot solve all anonymity problems. A network adversary able to see traffic from the client to guard and from the exit to destination can launch a correlation attack that could use statistical analysis to discover if they are part of the same traffic. We can then correlate these two traffics together and say that this specific client is going on this specific server.

2.2 BGP: Border Gateway Protocol

The Internet is a connection between multiple Autonomous Systems. An AS is a computer network owned by a single administrative entity. It is typically an Internet service provider (ISP) or a very large organisation. Each AS is given IP prefix(es) that will be used to assign IP addresses to all the devices inside their network. The distribution of prefixes is done by the Internet Assigned Numbers Authority (IANA) which is an organization that manages the Internet resources. The IANA notably splits the IP address space into the five Regional Internet Registries (RIR) and they will then split their IP address space with the ASes inside their territory.

When an AS has to send a packet to a certain IP address, it has to know the path towards it. In order to solve this, BGP exchange path towards prefix of ASes. When an AS has to send a packet to an IP address, it will look inside his BGP table to see at which AS it has to send the packet. The AS that receives the packet will do the same until the packet arrives inside the right AS. Here the packet will be forwarded inside the AS towards the right device.

ASes communicate with each other with BGP connections. This connection is above the TCP layer on port 179. On this link they can exchange multiple messages [8]:

- OPEN: To begin a connection and exchange AS number, Router ID and network capacity.
- KEEPALIVE: Message sent every 30 seconds to maintain the connection. After 90 sec without KEEPALIVE or UPDATE messages, the connection is closed.
- UPDATE: Announce or withdraw IP prefixes.
- NOTIFICATION: Message after the end of a BGP session after an error.
- ROUTE-REFRESH: Message to ask/reannounce prefixes after a modification of the routing policy.

To understand the propagation of an UPDATE message, we can look at figure 2.3. AS1 wants to announce the prefix 2001:db8:cafe::/48. It sends an UPDATE message to AS2 and AS4 with the AS Path: AS1 (this means the prefix is from AS1). AS4 will send the UPDATE to his only neighbor AS2. Now AS2 received two UPDATES with the same prefix but with two different paths. To decide which path to choose, it will refer to his BGP process. AS2 will prefer the route from AS4 and announce it to AS5.

The BGP process chooses a path between multiple paths of the same prefix by looking at a hierarchical list of preferences [8].

1. The AS will prefer the path with the biggest LOCAL_PREF. Each AS has financial relation with its AS neighborhood. It can be a relation of:

    - Customer to provider. This means all the traffic going to the provider will be taxed and you will have to pay.
    - Provider to customer. You will gain money when the customer sends traffic to the provider.

- Peer to peer. The traffic exchange is free in both directions.

  As you can imagine, an AS will firstly chooses the path announced by customers then by a peer, and finally, by a provider. LOCAL_PREF is a value that is used to provide this service.

2. The AS will prefer the smallest AS_PATH. Often, the smallest is the path, the fastest the traffic will arrive at the destination. Each time an AS will have to forward an announcement, it will add it AS number at the end of the AS_PATH.

3. The AS will prefer the prefix announced with the smallest MULTI_EXIT_DISC. An AS can have multiple BGP connections with other AS. ASes in this configuration will send the BGP UPDATE messages on all links. The AS can add a value, called MED, to these messages to tell the other AS which connection is preferred to receive the traffic.

Others preferences exist but these are the main ones.

As AS2 chooses the path announced by AS4 in figure 2.3, we can consider that the LOCAL_PREF from AS4 is bigger than the one from AS1. Indeed, the AS_PATH in the prefix announcement by AS1 is smaller than the one by AS4. AS1 seems more attractive but, as it is mentioned above, LOCAL_PREF is higher in the BGP decision process. AS4 is certainly the customer of AS2.

When an AS has to forward packets, it will look in his BGP table and choose the most specific prefix based on the destination IP address. It will then send the packets towards the best path chosen.

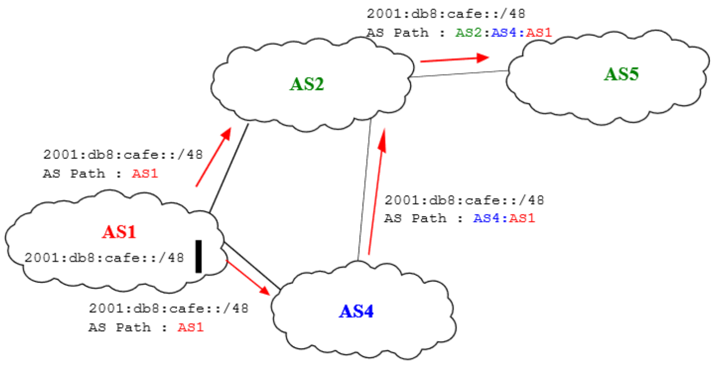


Figure 2.3: Example of how a prefix is announced by an AS. The image is from [9]

CHAPTER 3

State of the art

Tor is known to be vulnerable to attackers that can see traffic from both ends of the circuit. In this situation, we are then able to proceed a correlation attack. This will lead to a drop of the user's anonymity.
In this chapter, we will summarize papers made by researchers related to this subject. It will be divided into three sections.
First, we will look at attacks to reroute traffics to be able to observe them.
Then we will explain how to perform correlation attacks in practice.
Finally, we will see multiple defenses on this problem. Two families of defenses exist for this attack. The first one is to reduce the possibility of an attacker to observe both sides of a traffic. The second is to try making impossible correlation attacks even if you can see both sides of the traffic.

3.1 Attacks to observe traffic from both sides of Tor network

To be able to watch the two sides of the traffic, you can just be lucky and be on the path of both of them. This is unfortunately not always the case. A suite of three attacks launchable by ASes called RAPTOR has been found to do this job [3]. These attacks exploit the asymmetric and dynamic properties of the Internet.

3.1.1 Asymmetric Traffic Analysis

When you establish a communication with a server on the Internet, the path used to reach the server is not necessarily the path used by the server to reach you. This will increase the number of AS able to observe this traffic.
Since the communication in Tor uses TCP, there is therefore a flow of TCP sequences and a flow of TCP acknowledgments in the network. Moreover, there are four states in which ASes can see any direction of the traffic from both sides. They have access to:

- Data traffic from the client to the guard and data traffic from the exit to the server.
- Data traffic from the client to the guard and acknowledgment from the server to the exit.
- Acknowledgment from the guard to the client and data from the exit to the server.
- Acknowledgment from the guard to the client and acknowledgment from the server to the exit.

If an AS is in one of these situations, it can perform an asymmetric traffic analysis. This analysis is described below.

To illustrate the concept, you can see in the figure 3.1 all flows of the communication. AS3, AS4 and AS5 are able to launch the attack.

3.1.2 Natural Churn

As the Internet does not have a static structure, the physical links of the network can break or others can be created and the AS-level routing policies can change. All these changes will cause modifications to the path taken by packets.
A malicious AS could make a link failure to change the path of traffic. This can increase the number of AS able to observe this traffic. In figure 3.2, AS4 breaks the link with AS5 to let AS3 observes the traffic as well.

3.1.3 BGP Hijack and Interception

The Border Gateway Protocol is a protocol which advertises the set of IP prefixes that each AS owns and to forward theses prefixes to all ASes. Unfortunately, BGP has not been developed to face wrong announcements over the network. A malicious AS can announce other IP prefixes that do not belong to it (BGP hijack)

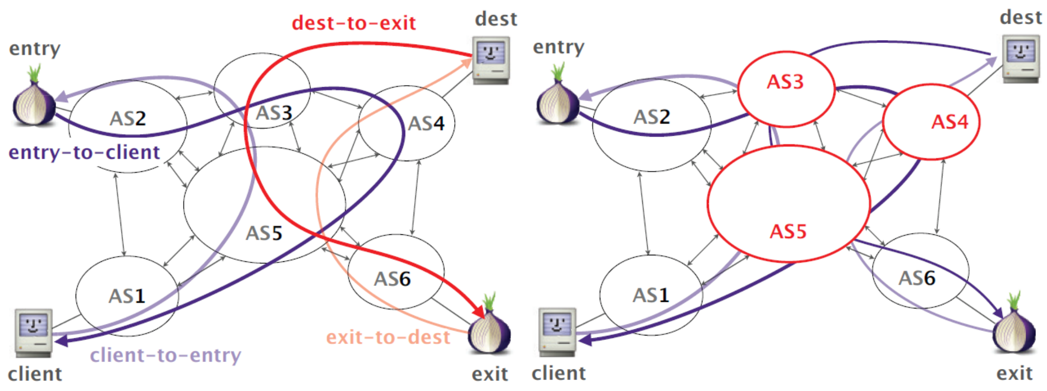


Figure 3.1: The asymmetric property of the Internet makes more ASes see any direction of the traffic at both ends of the communication. The image is taken from [3]

or forge AS paths (BGP interception). BGP hijacking has been a problem for over 25 years.
If a prefix is hijacked, we can blackhole the traffic, intercept the traffic, impersonate the legitimate receiver of the traffic or use the prefix for spamming [10]. We will look at blackholing and intercepting the traffic.

When a malicious AS does a BGP hijack, it advertises over the Internet a foreign prefix of his own. This will split the Internet about whom as got this prefix. Each AS that obtains the BGP advertisement will use their BGP process to decide what is the best route. All traffic in the range of this prefix towards the malicious AS will be blackholed because the malicious AS will not be able to forward the traffic to the right destination. Figure 3.3 shows an example of a BGP hijack.

To avoid the drop of the connection as in BGP hijacks, the malicious AS can forge an AS path with his ASN (AS Number) in it to be on the path. Figure 3.4 illustrates this mechanism. Instead of advertising the prefix 10.0.0.0/16 with AS path:3 (BGP hijack), it announces the prefix with his ASN in the AS path with a real path towards the real source AS of this prefix. The connection will then be maintained during communication.

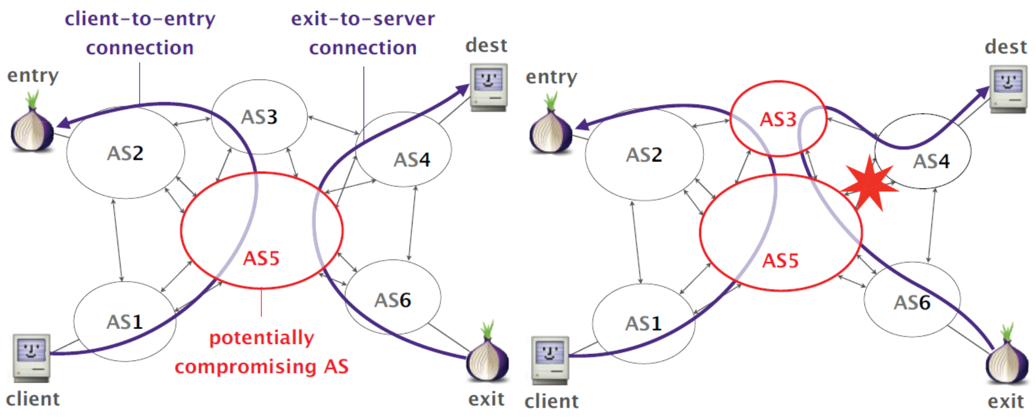

client-to-entry connection
exit-to-server connection
dest
entry
AS3
AS2
AS4
AS5
AS6
AS1
potentially compromising AS
client
exit
dest
entry
AS3
AS2
AS4
AS5
AS6
AS1
client
exit

Figure 3.2: AS4 stops the BGP connection with AS5. This will reroute the traffic to AS3 and finally to AS4. AS3 is now on the path of this flow. The image is taken from [3]

3.2 Correlation attacks in practice

Correlation attacks can be of multiple forms. It can simply be a correlation of two symmetric TCP flows or a correlation of two asymmetric TCP flows. Furthermore, other research has demonstrated that it was possible to make correlation attacks with TCP flow from ingress traffic and DNS flow from egress traffic.

3.2.1 Correlation attack with two symmetric TCP flows

In this situation, we can observe both directions of the TCP traffic. We can perform a correlation attack by analyzing the packet sizes and/or packet timings [11]. Another possibility is to match TCP sequence number and TCP acknowledgment number with ingress and egress traffic [3].

To decrease the false positive, we can also use the technique of watermarking. This approach tries to make a particular flow different from the inherent pattern of other network flows. A watermark encodes only one bit of information in the traffic that will be easily checked in other potential same traffics. [12].

3.2.2 Correlation attack with two asymmetric TCP flows

This attack retrieves the TCP sequence number and/or the TCP acknowledgment number of all packets observed in client-side connections and server-side connections.

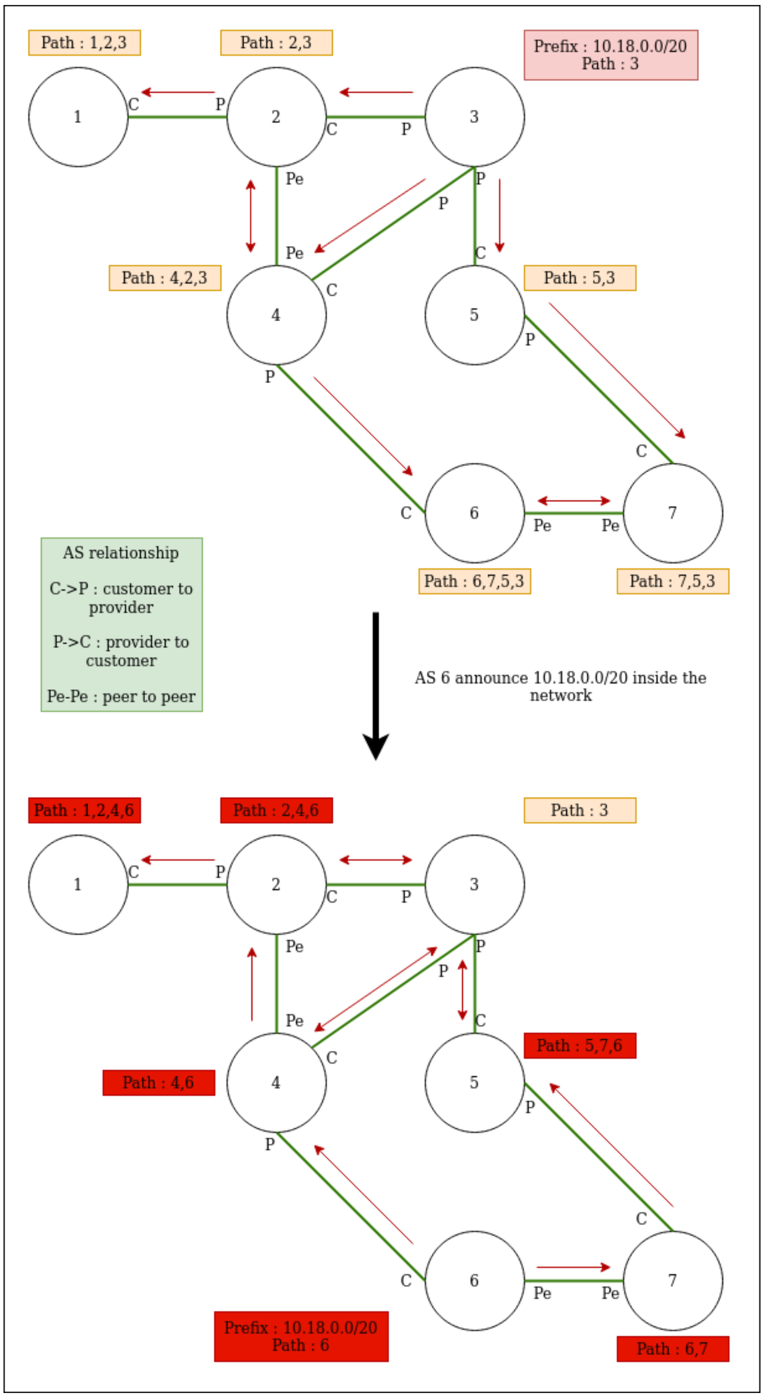


Figure 3.3: Example of BGP hijack with BGP process using only AS relationship and AS Path. AS 3 announces 10.18.0.0/20 to the network. The first part shows the state of the network after the announcement. We simulated the BGP hijack of the prefix by AS 6. The second part is the state of the network after the propagation of the new path. All the network except AS 3 are tricked.

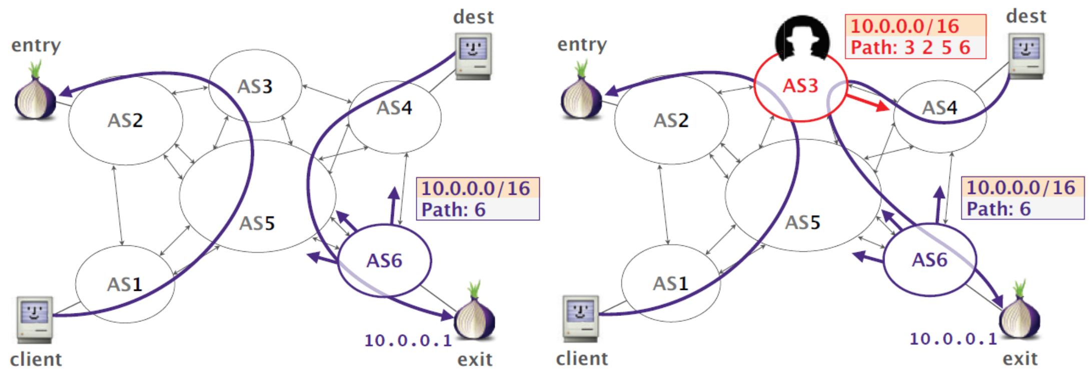


Figure 3.4: Initially AS3 only sees forward traffic. By announcing the prefix to AS4 with better properties for the AS4 BGP process, this path is used. AS3 is now on the path. The image is taken from [3]

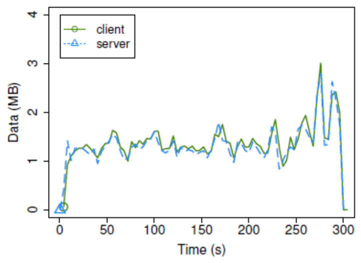
The number of transmitted data bytes per unit of time is then computed. Finally, for each pair of observed traces from each sides of the Tor network, we compute a correlation between data transmitted in the time (Figure 3.5).

3.2.3 Correlation attack with ingress TCP and egress DNS

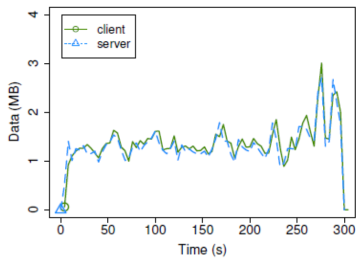
This attack is different from the previous correlation attacks and we will not go deep in detail as this is not in the scope of this thesis. It relies on the possibility to perform a website fingerprint attack on the ingress traffic in order to discover with a certain success rate the website visited. On the egress traffic, we will analyze the DNS packets to find the destination server. With these two pieces of information, we can perform a correlation attack between the two websites discovered from each edge of the Tor network. It is called a DefecTor attack and it is represented in figure 3.6. [13]

Website fingerprint attack is made by analyzing the packet's inter-arrival timings, directions, and frequency. An attacker can match this information with a database created earlier containing the same information of a multitude of websites.[14][13]

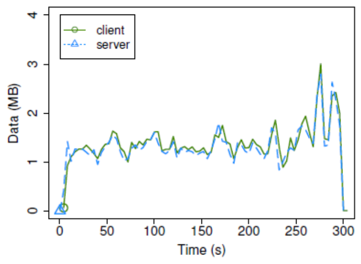
The same procedure can be done for the DNS part. For a set of domains found on Alexa (list of top websites), we save all the DNS requests to access the website. We can now correlate DNS traffic and websites.[13]



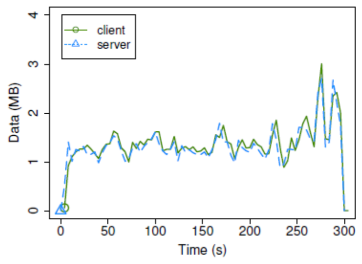
(a) Client: ACK, Server: ACK



(b) Client: ACK, Server: Data



(c) Client: Data, Server: ACK



(d) Client: Data, Server: Data

Figure 3.5: These graphs show the high correlation between a matched client-side and server-side traffic. The image is taken from [3]

3.3 Defense of correlation attacks

3.3.1 Taxonomy of countermeasure

The countermeasures are mentioned in figure 3.7. As Tor is oriented to provide a service with low latency, the mitigation of correlation attacks is not a good path to follow. Obfuscating packet timing/size or encrypt TCP header/randomize TCP Ack protocol will be too costly for the Tor properties. Moreover, going this way will increase the latency of the connection passing through the Tor network. This is true for several reasons.

- Obfuscating packet timing: We will not send all the packets directly outside the Tor network which will increase the latency.

- Obfuscating packet size/encrypt TCP header: Manipulating the packets will consume more time than sending them directly which will increase the latency.

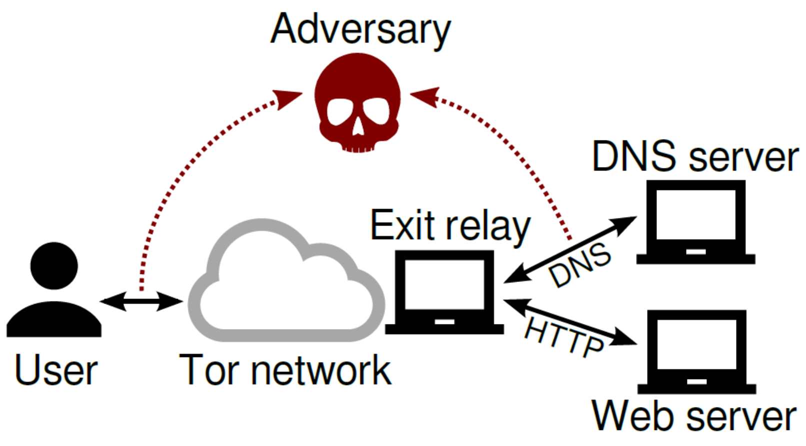


Figure 3.6: Illustration of the DefecTor attack. The image is taken from [13]

3.3.2 Secure BGP deployment: RPKI

The best way to counter BGP attacks would be to fix the problem at the source. BGP is not a secure protocol. As a matter of fact, anyone who wants to announce a prefix not owned can do it. This leads to severe consequences as hijack is not a wanted behavior. Several propositions have been made to upgrade BGP to make it a secure protocol. The Resource Public Key Infrastructure (RPKI) is the most promising one.

RPKI provides signed data about which AS is authorized to announce which prefixes. In more detail, each RIR (Regional Internet registry) provides a portal where you can manage your ASN and your IP prefixes. On this portal, you can create a ROA (Route Origin Authorization). This is a certificate signed by the RIR about your ASN and the IP prefixes that you have. All the ROA are shared over the five RIR so everyone has access to all ROA on the Internet. Anyone can ask the validator of their RIR if a BGP update has an origin AS from the right AS.[15]

Unfortunately, this solution is not widely adopted by ASes. In 2018, only 10% of the ASes had deployed RPKI solution on the Internet. Moreover, 32% of the Tor network is under RPKI but 12% of the prefixes have a weak max length in their ROA. [16]

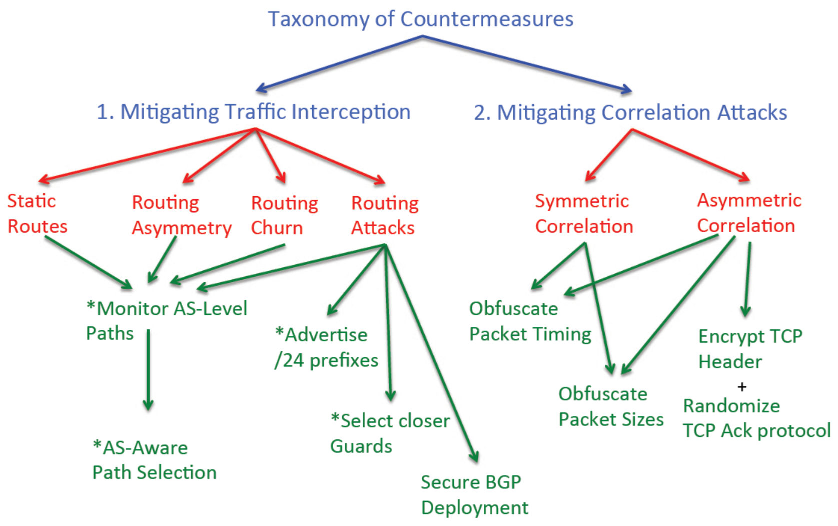


Figure 3.7: Map of possible countermeasures. The image is taken from [3]

3.3.3 Tor guard relay selection aware of active BGP Hijack

During a BGP hijack of a more specific prefix, all the entities of the network will prefer the path toward the malicious AS. If the prefix announced is equally-specific, this will split the Internet into two. On the one hand, a set of ASes will prefer the path toward the malicious AS, on the other hand, another set of ASes will prefer the path toward the true origin AS. By choosing a guard relay wisely, the Tor client can stay unaffected during such attacks.

The guard relay selection algorithm could choose a circuit without passing through the same AS twice. However, this is not sufficient as a BGP hijack or interception can occur after the creation of the circuit.

If a malicious AS launches a BGP hijack in the range of a Tor Guard relay, all the AS that prefer this route will forward packets towards the false origin AS. The malicious AS can then see all IP and ends of connections of all users of this Tor guard relay.

To resolve this issue, we can compute the resilience of all ASes containing Tor relay on BGP hijack. The client will then also consider the resilience of the AS containing the guard during the relay selection algorithm.[4]

Explanation of resilience

The concept of resilience is defined in [4]: a source AS **v** is resilient to a hijack attack launched by a false origin AS **a** on a true origin AS **t**, if **v** is not deceived by **a** and still sends its traffic to **t**.
Three computable resiliences exist:

- Origin-source-attacker resilience: The probability of AS **v** being resilient to a given source AS **t** and attacker AS **a**.
- Origin-source resilience: The probability of AS **v** being resilient to a given source AS **t** and all AS can be attackers.
- Origin resilience: The probability of AS **v** being resilient to all situation where all AS can be sources and all AS can be attackers.

The three types of resilience are linked. To compute the origin-source resilience, we have to know all the origin-source-attacker resilience. To compute the origin resilience, we have to know all the origin-source resilience.

To calculate the Origin-source-attacker resilience given (**t**,**v**,**a**), malicious AS **a** will hijack a prefix owned by AS **t**. If AS **v** chooses the path to **a** then the resilience is 0. Otherwise, if AS **v** chooses the path to **t** then the resilience is 1. If the paths are equal in the BGP process, we compute the resilience as $\beta(t, v, a) = \frac{p(v,t)}{p(v,t)+p(v,a)}$, where p(v,a) is the total number of paths to malicious AS **a** and p(v,t) is the total number of paths to AS **t**.

To calculate the Origin-source resilience, we add all the origin-source-attacker resilience from the same origin and source together. Then we divide it by the number of ASes minus 2 because the origin AS and Source AS are not attackers.

To calculate the Origin resilience, we add all the origin-source resilience for all sources then we divide it by the number of ASes minus 1 because the origin is not a source. In the case of this counter defense, we only compute this resilience for origin AS containing Tor relay.

Simulate BGP hijack

Now that we know how to compute the resilience we need to figure out how we can simulate the Internet and create BGP hijacks.

First, we will look inside the Tor consensus archives to extract the IP addresses of all guard relay from the Tor network. The document has a part showing the relays information and tells if the relay is a guard with their corresponding IP address [17].

Then, we have to create a virtual version of the Internet by using BGP archives and AS relation archives. We will create a virtual network in which nodes will be AS and edges will be a link between these ASes. The BGP archives will tell us the link between ASes by analyzing the BGP path. Indeed if the AS path is [125, 456789, 3216, 456, 78952], we know that all these AS are linked to their neighbors. We will also use the AS relation archives to complete the link between ASes. Indeed, if AS 456 and AS 789 as a relation of peer-to-peer we know that there are linked to each other. Our virtual network is created.

We will now fill the BGP table inside each AS of the network. The BGP archives will tell us which AS is announcing which prefix. We will filter them to only take announcements from Tor guard relays. After having all the prefixes and the corresponding AS, we will propagate them inside our virtual network. The BGP process done during the propagation of the prefixes will use the AS relation and the AS path. Indeed, an AS will prefer a prefix announce by a customer then by a peer, and finally by a provider. If there are multiple paths towards best AS relation, we will choose the shortest AS path toward the prefix. These two preferences will mainly determine the AS paths chosen by ASes [18]. We have now the topology and the BGP tables of ASes.

We can now simulate BGP hijack in the virtual network and get the origin resilience.

3.3.4 Monitoring of BGP hijack over the Internet

All the defenses seen previously are proactive. This means that they try to defend Tor before the attack occurs. In [4], the authors also follow a reactive approach. They propose a system that monitors suspicious BGP activities for routing attacks that could affect Tor relays

This defense works in two phases.

1. Collecting Monitoring Data: obtain BGP data from multiple BGP collectors all around the world. For each announcement, we look if the prefix contains a Tor relay IP. For each of these announcements, we make an IP to ASN mapping to know the true origin AS.
2. Detecting Routing Anomalies (suspicious behavior):
    - If the origin AS in the data is not the same as the one from the IP to ASN mapping.

- An attack is a prefix not owned and announced once. If the frequency of the announcement is below some threshold.
- An attack stays a relatively short amount of time in comparison to a normal BGP announcement. If the time of life of the path is below some threshold.

The prefixes in suspicious updates are blacklisted and the guard relay having an IP address in the range of the prefix shall not be used as guards for some time.

CHAPTER 4

Motivation for the implementation

As explained in the introduction, we would like to compute security metrics about BGP hijacks in Tor. These metrics will be of two types:

- The resilience of guard and exit relays from the Tor network on BGP hijack. Each relay will be given a resilience each month.
- A list of potential BGP hijacks sorted chronologically.

With these two metrics, we can visualize more easily the current robustness of Tor against BGP hijacks.

In [4], the authors already proposed solutions but they only computed resilience of the 1$^{\text{st}}$ January 2016 and monitor BGP for four months (February to May 2016).

In the same paper, they only computed the resilience of the guard Tor relay. They did this because they wanted to improve the Tor guard selection algorithm. In this master thesis, we wanted on the other hand to see the how much the network is robust on BGP hijack. As correlation attack uses traffic going to the guard relay and uses traffic going out the exit relay, these relays were decided to be taken in account.

In this master thesis, we also want to go further by giving tools that could compute these metrics from the beginning of Tor to live time.

CHAPTER 5

Implementation of resilience computation program

5.1 Structure of the program

In this program, the resilience of Tor relays on BGP hijack will be computed. This means the proportion of ASes that will choose the path to the right origin when a malicious AS announces the IP prefix containing the Tor relay.

A virtual network will be created to represent the Internet. Networkx, an API to create networks, was chosen. Each node is an AS and each edge is a link between ASes. Finally, each node contains a BGP table with the route that it has learned.

Each day the BGP announcement containing Tor prefixes will be extracted. These announcements will be propagated inside the virtual network.

Now that the network is up-to-date for the day, the resilience of all Tor relay IP prefixes on BGP hijack will be computed as explained in the state of the art. However, some parts of this process were modified for performance reasons.

Link to project repository

The code can be found in the following Github repository :
https://github.com/Wvisee/RAPTOR.git

5.1.1 Schema resuming the program

For a better understanding of the program, the figure 5.1 shows how the program works.

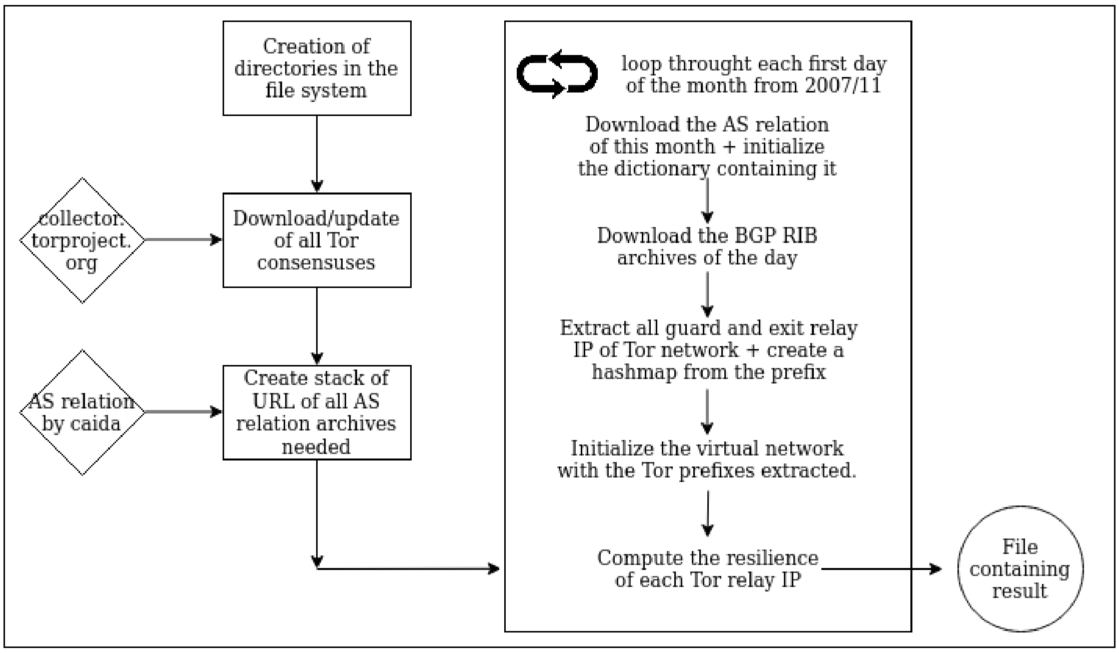

Creation of directories in the file system
collector. torproject. org
Download/update of all Tor consensuses
AS relation by caida
Create stack of URL of all AS relation archives needed
loop throught each first day of the month from 2007/11
Download the AS relation of this month + initialize the dictionary containing it
Download the BGP RIB archives of the day
Extract all guard and exit relay IP of Tor network + create a hashmap from the prefix
Initialize the virtual network with the Tor prefixes extracted.
Compute the resilience of each Tor relay IP
File containing result

Figure 5.1: Summary of how the resilience program works

5.1.2 Input

The program will use several external services.

1. Tor consensuses from Tor collector [19]: It contains all the information about the Tor network. The IP addresses of the guard and exit relays will be extracted from these files.

2. Ripe NCC [20]: A service having BGP collectors all around the world catching BGP announcement. The RIB tables available will be used.

3. AS relation from Caida [21]: It contains all the information about the BGP relation between ASes. It is all the relation of provider, customer, and peer as explained in the background.

4. Mrt2bgpdump by t2mune [22]: All the BGP archives used in the program are compressed in Multi-Threaded Routing Toolkit (MRT) format. To

analyze them, this format will be transformed using Mrt2bgpdump into the "Bgpdump" format.

5.1.3 Program in detail

Step A. Create directories in the file system that will contain data needed to make the computations.

1. tor-consensuses-tar: Contains all the archives of the Tor network compressed in Tar. A file named last_changed, containing the date of the last modification of each consensus, will also be created.
2. tor-consensuses: Contains the current month of Tor network data. It will be updated during the run of the program.
3. rib: Contains the current RIB archives of the day being analyzed. It will be updated during the run of the program.
4. Programs: Contains a python script to extract data from BGP archives in MRT format.
5. Tmp: Contain all the temporary files to be analyzed. Most of them are HTML files.
6. Result: Contains the resilience of all Tor relay the first day of each month. It will be updated during the execution of the program.

Rib and Tmp will be cleaned at each iteration of the program. The other files mentioned above will not be deleted as they are required for the good program's operation.

Step B. Download the Tor network consensuses of each month into the directory "tor-consensuses-tar". The directory will be updated if consensuses are already downloaded. In more detail, the Tor collector website [19] provides us the URL of the archives by checking the code source of the HTML page and the archives will be downloaded recursively. If an archive is already downloaded, the server version is checked and the latest version is downloaded if needed.

Step C. Create a sorted URL stack of the AS relation archives. Each URL contains an archive with data of one month of AS relation. Caida provides this data [21]. Each archive from each month will be downloaded during the main process.

Step D. Iterate through all the Tor network consensuses. The archives will be untared chronologically. At each consensus, an iteration will be made through the day of the month. On each first day of the month, the main analysis will be launched.

Step E. The main analysis on each first day of the month is described here.

1. Download the AS relation of the current month and initialize the dictionary containing this information.

2. Download all the RIB archives of the current day (the first day of the current month). Only the archives of Ripe NCC will be downloaded as the number of archives is big. The Tor prefixes will only be extracted from the tables of the collectors. The Tor prefixes announced on the Internet can be supposed to be at least in the Ripe NCC collector. Looking at collectors from other services will slow down the program without giving new prefixes.

3. Extract IP addresses of the guard and exit relay of the Tor consensus of the current day. The 24 Tor consensus of the day will have to be filtered. An explanation on how consensus are structured can be found on Jordan Wright's blog [17]. Each router has a bunch of flags. The r flag will give us the IP address of the relay and the s flag will tell us if the relay is guard or exit.

4. Build a hashmap with all the IP prefixes from /17 to /24 that match their IP address. It will be necessary to filter the further RIB tables to know if the prefix inside a table is containing a Tor relay IP address.

5. Extract all the entries containing Tor prefixes inside the RIB tables downloaded. The ASN and the prefix announced will be saved.

6. Propagate all BGP announcements obtained inside our virtual network. In detail, all the ASes and the prefixes extracted will be sent inside the virtual network and the BGP process of all the AS inside the virtual network will be simulated. The BGP process will use the AS relation and the AS path as the input of the process.

7. Our virtual network is now up-to-date for the day. The resilience of each Tor relay on BGP hijack will be computed.
   As explained in the state of the art, the Origin resilience for each Tor prefix will be computed. In practice, this process is long so the number of Origin-source-attacker resilience computed was reduced. It is explained in the design choice part.

8. Delete the RIB archives from this day and clean the Tmp directory for the next computation.

5.1.4 Output

The output of this program will be written in a file. At each loop, the date will be written (year-month-day). Then, all the Tor relay IP addresses inside the virtual network, their attached AS, prefix, and the resilience computed, will be written down in the file.

5.1.5 Dependencies

The program is written in Python3. It has only been tested on Ubuntu 18.04 [23]. It will need to run Bash as several Bash commands are used in the program. The API list is as follow: os, urllib, fileinput, sys, datetime, time, socket, networkx, pickle, errno, copy, gc. An Internet connection is also required.

5.1.6 Setup the OS

As the program is taking a lot of resources, the Operating System (OS) sometimes kills the process. There exists, in Linux, a program called the Out of memory killer[24] (OOM killer) that will kill process allocating too many resources. To protect our process from being killed during its execution, the OOM killer will be told not to kill this process.
Entering the following command inside a terminal will do it:

```
$ echo -17 > /proc/$(pid of resilience program)/oom_adj
```

The "-17" is telling OOM not to kill the wanted process.

5.2 Design choice

5.2.1 Resilience with an error rate

The computation of the origin-resilience is exponential with the number of AS. To compute all the BGP hijack scenarios, each AS will be the attacker and source. This is a lot of computations. In order to have a more reasonable time of execution, increasing the error rate by decreasing the number of attackers was a good compromise. The value was set to 100.

5.2.2 Use of Tor collector instead of Tor API

The Tor consensuses inside our program are used to get the IP addresses of the guard and exit relay. There are two ways of getting them. The Tor API or the Tor collector website could be used. Both solutions have been tried during the programming of this part. They both have pros and cons.

1. Tor collector website: It was a more complex program to code because it has to download the HTML pages of the website to filter the useful URL toward the TOR consensus. It is a lot of condition (everything is in the code) but the best thing is that the archives were all inside my computer. If the program needed to run a second time, it did not need to download the files a second time.

2. Tor API: It was a simpler program. Fewer than five lines to loop through all the consensuses of the Tor network. Nevertheless, the problem was that, at each execution, it was needed to download all the consensus again. Compared to the first solution, it was costing more time.

The Tor collector website was chosen as it was costing less time to make the program run more than once.

5.2.3 Only filter BGP announcement with /17 to /24

During the filtering of the BGP RIB tables, the prefixes inside it had to contain IP addresses from a Tor guard/exit relay. To do this, a hashmap was created with all the prefix possible from all the IP addresses of guard/exit relay inside the current Tor network. This means for example that if the network contains 1000 guard/exit relays, there will be about 32000 prefixes inside the hashmap. This solution makes the process faster to know if a prefix is from a Tor relay. The complexity is O(1).

Surprisingly, ::/0 to ::/16 prefixes was being announced on the Internet. As there were a lot of them, the program was really slow as these RIB entries were used during the propagation phase. As BGP normally does not care of so small prefixes, our program does not take them into account.

5.2.4 No IPv6

Initially, IPv6 prefixes were taken into account but when the hashmap of all the prefixes from a bunch of IPv6 addresses was made, the hashmap was already huge and this was too long to make. Indeed, an IPv6 address is 128 bits long. The insertion of all the prefixes from /12 to /64 (the IPv6 prefix used in BGP [25]) inside a hashmap was creating a too big hashmap.

The other possibility was to look if the IPv6 address inside the RIB tables was an IPv6 address from a Tor guard/exit relay by doing the computation during the process but this was slowing the program down too much.

Besides, using IPv6 considered announcing more prefixes inside our virtual network for the next step. To speed up the program, IPv6 prefixes were not taken into account.

5.2.5 Choice of the BGP process when two paths are equivalent

During the propagation of the prefixes inside the virtual network, the BGP process of all the ASes inside it had to be simulated. The BGP process will take into account the financial relation between ASes as well as the length of the path. These are the two main preferences done by BGP process in the real world [18].

Sometimes the BGP process has to choose between two paths with equivalent AS relation and path's length. The new path is always chosen. This choice will not impact the initialization of the network. Meanwhile, during the resilience computation, a prefix will be propagated from random ASes. This new path will be forced to trick the BGP process of the AS. The worst scenario is chosen to evaluate the biggest potential threat.

5.3 Limitations

5.3.1 Speed

Downloading and going through all the BGP RIB archives is long as it is a huge quantity of data. The program will pass about 50% of this time downloading and extracting the announcement about Tor prefixes.

During the first version of the program, it downloaded the RIB archives from Ripe NCC and Routeview. The quantity of archives was to big to run the program in an reasonable amount of time. So only one service was used: Ripe NCC. For the consistency of the prefix extracted this service is supposed to catch all the prefixes announced on the Internet.

5.3.2 Memory usage

Considering the size of the virtual network and the number of prefixes to announce inside it, a certain amount of memory will be used. If these inputs are too big, the program can crash even with OOM disable[24]. The resilience was computed until 2015. From 2016, the Internet and Tor are too big to make the computation with 8GB of RAM.

5.4 Further improvements

5.4.1 Daily computation of resilience

The program was slower and too greedy than expected. As a consequence, the result proposed could not be computed. Only the first day of the month of November from 2007 to 2015 was computed. Initially, the resilience of Tor relay IP prefix each day was expected.

5.4.2 What if the entry in the RIB table is a BGP hijack?

When a prefix is being extracted inside a RIB table, the AS that announced this prefix is not being checked if it is legitimate to announce it. An improvement would have been to use an ASN to IP service like the one of team Cymru [26]. If the origin check is saying that it is not the right AS to announce this prefix, a note could be written next to the resilience score inside the output. The tool is not perfect so ignoring an ASN-prefix could be a data loss.

5.5 Issues encountered

During the design and the writing of the program, issues were encountered. These issues were slowing a lot the execution of the program. Some of them took some time to be reduced.

5.5.1 Use of BGP updates from all BGP collectors

Before using the BGP RIB archives, the BGP updates archives were used. These files contain BGP announcements.

The first problem was to get all these files. Storing all the files on my computer was expected but the quantity of data was too big. Storing all the URL of archives from Ripe NCC and Routeview sorted chronologically was then decided. Each day, the top of the stack would be popped until the URL of archives of the next day. This stack was long to compute. It takes around one hour and the stack was around 1GB. As the program takes more time than expected, this step was done before the main loop to speed up a little bit the execution. Nevertheless, it was still time-consuming.

The second problem was that sometimes the program crashed due to the quantity of memory used. This was a problem as the virtual network was being reset. To have the same state of the network, all the computations from the beginning had to be remade. A solution could have been to pickle the network each day, but it was more overhead.

To solve this, two improvements were deployed.

1. Using less archives so only Ripe NCC archives were used. Fewer archives to pass thought, less time to execute. In addition, it was not faking the result as Ripe NCC had BGP tables containing the prefixes from all Tor relays normally.

2. RIB tables were used instead of BGP updates to delete the crash problem. Indeed, if the program crashed The RIB tables of the day needed to be loaded and the computation could be continued.

5.5.2 Bad configuration of the BGP tables inside the virtual network

When the program was finished, it was not fast enough to compute all the wanted results in the given time. The bottleneck was a function returning a list of the best path from a list of paths toward a prefix considering only the relation between ASes. The first idea was to optimize it. The final version of the function is shown in figure 5.2. It helped dividing the time by two but it was still not as fast as expected.

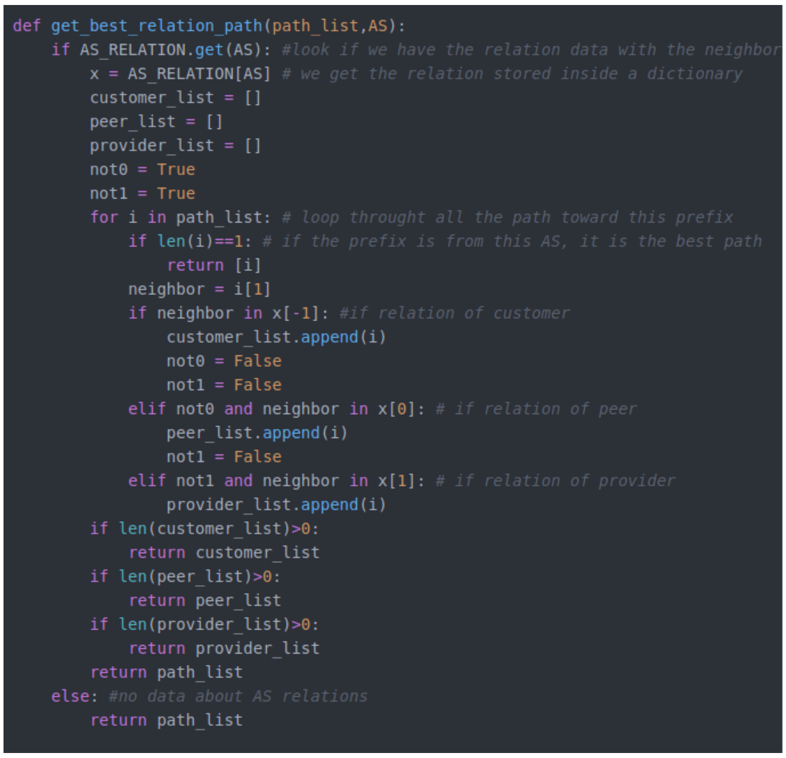
The only variable was the size of the list containing the paths toward the same prefix. What if this list was reduced? The issue was that the BGP tables were not storing effectively the BGP announcement. When a prefix was propagated inside the virtual network, the Ases were storing all the paths. The best path and the other paths. In fact, only the best path needs to be stored and the other ones could be deleted. Indeed the worth paths will always stay the worth paths inside a static virtual network (The topology of the network will change from day to day).

The program was slow because, during the BGP process, it has to go through all the path each time.

The final code stores only one path toward a prefix by AS. When a new BGP announcement arrives at the AS, the BGP process will choose to keep the old path or change it by the new path considering the context.

5.5.3 Wrong copy of the BGP tables

During the computation of a resilience, the prefix is propagated several times inside the virtual network from several false origins AS. From all this results the average



```python
def get_best_relation_path(path_list,AS):
    if AS_RELATION.get(AS): #look if we have the relation data with the neighbor
        x = AS_RELATION[AS] # we get the relation stored inside a dictionary
        customer_list = []
        peer_list = []
        provider_list = []
        not0 = True
        not1 = True
        for i in path_list: # loop throught all the path toward this prefix
            if len(i)==1: # if the prefix is from this AS, it is the best path
                return [i]
            neighbor = i[1]
            if neighbor in x[-1]: #if relation of customer
                customer_list.append(i)
                not0 = False
                not1 = False
            elif not0 and neighbor in x[0]: # if relation of peer
                peer_list.append(i)
                not1 = False
            elif not1 and neighbor in x[1]: # if relation of provider
                provider_list.append(i)
        if len(customer_list)>0:
            return customer_list
        if len(peer_list)>0:
            return peer_list
        if len(provider_list)>0:
            return provider_list
        return path_list
    else: #no data about AS relations
        return path_list
```

Figure 5.2: Code of function telling the best path considering only AS relation

of these are calculated. The resilience for this prefix is now available.

In practice to do this computation, a copy of the BGP tables has to be made each time that a BGP hijack is simulated. Indeed, the clean state of the Internet must always be used before propagating a prefix.

The tables were stored inside a dictionary. The dictionary has a relation of AS to a second dictionary. The second dictionary has a relation of prefixes to a list of paths. This dictionary had to be copied but "copy_dict = dict.copy()" was used to do it but the function copy() only make a shadow copy of the dictionary. This means that the references are preserved[27]. As you can imagine the BGP tables were being changed with the BGP hijack and then the new BGP hijack was done on top of the previous BGP tables modified by the previous BGP hijack.

The right function to use is "copy.deepcopy(Name_of_dict)" from the copy API[27]. This will recursively make a copy of the dictionary. But instead of the deepcopy(), the dictionary was pickle[28]. This stored the dictionary on the drive which leaves space inside the RAM.

CHAPTER 6

Implementation of BGP hijack monitoring program

In this program, the BGP archives obtained over the Internet will be analyzed to find potential BGP hijack. The archives are accessed via the Route Views Project of the University of Oregon[29] and the Regional Internet registry RIPE NCC[20]. The BGP updates will be filtered with the one containing prefixes of Tor IP addresses. To discover the potential BGP hijack, an origin check first will be made. To make this possible, the IP to ASN mapping tool from team Cymru[26] will be used. If it is not the right AS announcing the prefix, the frequency of this announcement, and the time to live of the potential attack will be analyzed. An announcement of a prefix by another AS than the origin AS does not always mean that is it a BPG hijack. Suspicious BGP announcements are written into a file.

Link to project repository

The code can be found in the following Github repository :
https://github.com/Wvisee/RAPTOR.git

6.1 Structure of the program

6.1.1 Schema resuming the program

For a better understanding of the program, the figure 6.1 shows how the program work.

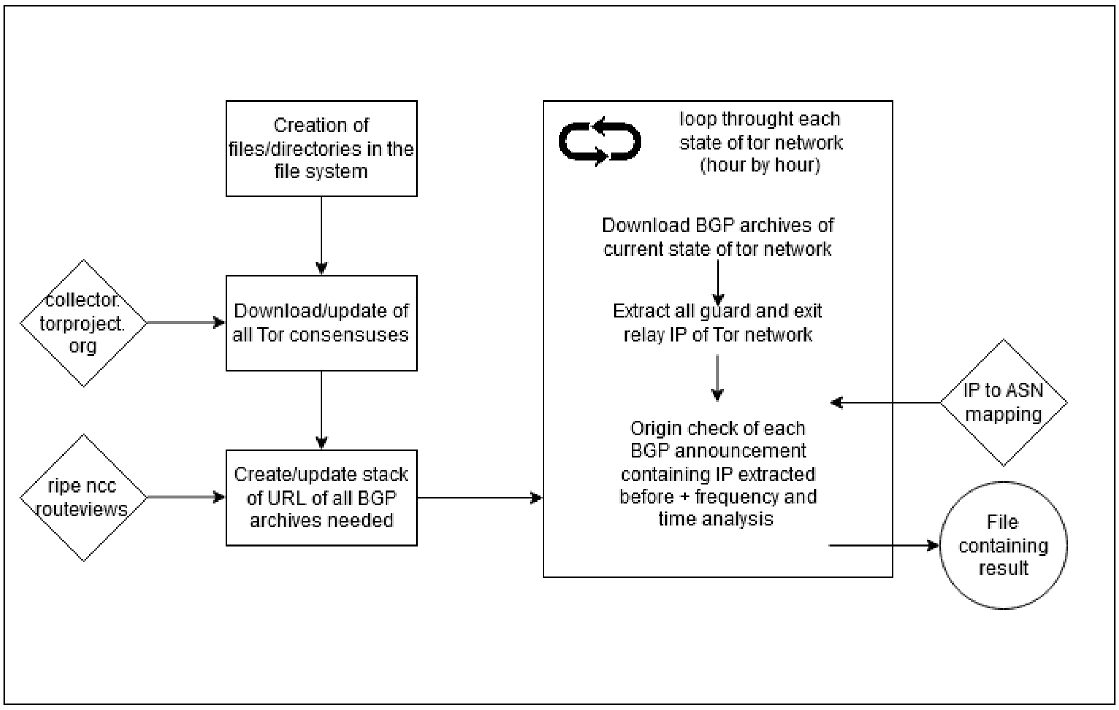

Creation of files/directories in the file system
collector. torproject. org
Download/update of all Tor consensuses
ripe ncc routeviews
Create/update stack of URL of all BGP archives needed
loop throught each state of tor network (hour by hour)
Download BGP archives of current state of tor network
Extract all guard and exit relay IP of Tor network
Origin check of each BGP announcement containing IP extracted before + frequency and time analysis
IP to ASN mapping
File containing result

Figure 6.1: Summary of how the BGP monitoring works

6.1.2 Input

The program will use several external services:

1. Tor consensuses from Tor collector [19]: It contains all the information about the Tor network. The IP addresses of the guard and exit relays will be extracted from these files.

2. BGP archives: It contains all the BGP updates captured by the BGP collectors from Routeviews [29] and Ripe NCC [20].

3. IP to ASN mapping tool from Team Cymru [26]: It is a tool that will return the ASN when giving an IP address.

4. Mrt2bgpdump by t2mune [22]: All the BGP archives used in the program are compressed in Multi-Threaded Routing Toolkit (MRT) format. To analyze them, this format will be transformed using Mrt2bgpdump into the "Bgpdump" format.

6.1.3 Program in detail

Step A. Create directories in the file system that will contain data needed to make the computations.

1. tor-consensuses-tar: Contains all the archives of the Tor network compressed in Tar. A file named last_changed, containing the date of the last modification of each consensus, will also be created.

2. tor-consensuses: Contains the current month of Tor network data. It will be updated during the run of the program.

3. BGP_Archives: Contains the current BGP archives of the hour being analyzed. It will be updated during the run of the program.

4. Data: Contains sorted lists of URLs chronologically of all the BGP archives that will be analyzed. The URL will be stored in the following files: BGP_url_stack_rcc, BGP_url_stack_routeview. A file called BGP_url_stack_routeview _history_download_archive will also be added that will contain the name of the collector and the date of the last update.

5. Programs: Contains a python script to extract data from BGP archives in MRT format.

6. Tmp: Contain all the temporary files to be analyzed. Most of them are HTML files.

7. Result: Contains the suspicious BGP announcements sorted by timestamp. It will be updated during the execution of the program.

BGP_Archives and Tmp will be cleaned at each iteration of the program. The files mentioned above will not be deleted as they are required for the good functioning of the program.

Step B. Download the Tor network consensuses of each month into the directory "tor-consensuses-tar". The directory will be updated if consensuses are already downloaded. In more detail, the Tor collector website [19] provides us the URL of the archives by checking the code source of the HTML page and the archives will be downloaded recursively. If an archive is already downloaded, the server version is checked and the latest version is downloaded if needed.

Step C. Create the sorted URL stack of BGP archives. The stack will be updated if it is already created. In more detail, the Ripe NCC BGP archives website [20] and on the Routeviews BGP archives website [29] provides all the URLs of all the archives from the 27th of October 2007, the beginning of Tor consensuses, to now. They will be sorted to be able to just pop the URL in O(1) during the analysis.

Step D. Iterate through all the Tor network consensuses. The archives will be untared chronologically. At each consensus, an iteration will be made through the day of the month. At each day, an iteration will be made through the hour of the day. Each hour, the main analysis will be launched.

Step E. The main analysis for each hour is described here.

1. Download all the BGP archives available of the hour from the current Tor consensus.

2. Extract IP addresses of the guard and exit relay of the Tor consensus of the current day. The 24 Tor consensus of the day will have to be filtered. An explanation on how consensus are structured can be found on Jordan Wright's blog [17]. Each router has a bunch of flags. The r flag will give us the IP address of the relay and the s flag will tell us if the relay is guard or exit.

3. Build a hashmap with all the IP prefixes from /17 to /24 that match the extracted IP addresses. It will be necessary to filter the further BGP announcements to know if the prefix inside the update is containing a Tor relay IP address.

4. "Untar" the BGP archives and loop through all the BGP announcements.

5. Catch only the update announcement with a Tor prefix.

6. An origin check is performed. If the AS announcing the prefix is not the expected one, it might be a BGP hijack. The frequency and the period of time of this suspicious announcement will be analyzed. If the frequency of announcement for this prefix is lower than 0.0025 (number of announcements by wrong AS/number of announcement by all AS) for the day it will declared as a BGP hijack. The threshold 0.0025 is taken from the counter RAPTOR paper [4]. If the period of time of this route toward this prefix is lower than one hour, it will also declare it as a BGP hijack. In detail, if a true origin AS is announcing the same prefix under one hour, this can be considered as a defense. The suspicious BGP hijack is now more suspicious and it will be recorded.

7. The analysis is now over. The BGP archives of the current hour are deleted. The process can be repeated.

6.1.4 Output

The output of this program will be written in a file. At each loop, the date and hour of the BGP archives will be written in the file. Then if likely BGP hijack is found, it will be written with the following structure:

1. Frequency: prefix announced: ASN: the number of times the prefix has been announced compared to all the other announcements for this prefix.
   example:
   Frequency : 195.199.250.0/23 : 35492 : 1/1180 = 0.000847457

2. Time: prefix announced: wrong announcer: right announcer: time in second between wrong announcement and right announcement.
   example:
   Time : 125.162.188.0/22 : 17974 : 7713 : 3593.0

6.1.5 Dependencies

The program is written in Python3. It has only been tested on Ubuntu 18.04 [23]. It will need to run Bash as several Bash commands are used in the program. The API list is as follow: os, urllib, fileinput, sys, datetime, time, socket. An Internet connection is also required.

6.2 Design choice

6.2.1 URL stack

Downloading all the BGP archives in local was the first idea but it was not going to be possible as the quantity of data was huge. A solution had to be found to simply download all the BGP archives from a slice of hours from multiple collectors and then do the analysis and finally deleting the downloaded files to get the next one. The solution of storing all the URLs of BGP archives sorted chronologically inside files was chosen. When the program starts the URL stack is created/updated and then transformed into a python stack. During the execution of the program, the stack is popped to obtain all the archives from the current slice of time.

6.2.2 Use of Tor collector instead of Tor API

The Tor consensuses inside our program are used to get the IP addresses of the guard and exit relay. There are two ways of getting them. The Tor API or the Tor collector website could be used. Both solutions were tested during the programming of this part. They both have pros and cons.

1. Tor collector website: It was a more complex program to code because it has to download the HTML pages of the website to filter the useful URL toward the TOR consensus. It is a lot of condition (everything is in the code) but the best thing is that the archives were all inside my computer. If the program needed to run a second time, it did not need to download the files a second time.

2. Tor API: It was a simpler program. Fewer than five lines to loop through all the consensuses of the Tor network. Nevertheless, the problem was that, at each execution, the consensus needed to be downloaded again. Compared to the first solution, it was costing more time.

The Tor collector website was chosen as it was costing less time to make the program run more than once.

6.2.3 Only filter BGP announcements with /17 to /24

During the filtering of the BGP archives, the prefixes inside it had to contain IP addresses from a Tor guard/exit relay. To do this, a hashmap was created with all the prefixes possible from all the IP addresses of the guard/exit relays inside the current Tor network. This means, for example, that if the network contains 1000 guard/exit relays, there will be about 32000 prefixes inside the hashmap. This solution makes the process faster to know if a prefix is from a Tor relay. The complexity is O(1).

Surprisingly, ::/0 to ::/16 prefixes were being announced on the Internet. As they were a lot of them, the program was really slow as these announcements were used during the propagation phase. As BGP normally does not care about so small prefixes, our program does not take them into account.

6.2.4 No IPv6

Initially, IPv6 prefixes were taken into account but when the hashmap of all the prefixes from a bunch of IPv6 addresses was made, the hashmap was already huge and this was too long to make. Indeed, an IPv6 address is 128 bits long. The insertion of all the prefix from /12 to /64 (the IPv6 prefix used in BGP [25]) inside a hashmap was creating a too big hashmap.

The other possibility was to look if the IPv6 address inside the BGP announcement was an IPv6 address from a Tor guard/exit relay by doing the computation during the process but this was slowing the program down too much.

Besides, using IPv6 was considering announcing more prefixes inside our virtual network for the next step. To speed up the program, IPv6 prefixes were not taken into account.

6.3 Limitations

6.3.1 Speed

The main limitation is that the number of BGP archives is huge. The program was launched for two months after the end of my programming and only three years of BGP data were analyzed. The data analyzed is from 27 October 2007 to 31 December 2010. The 27 October 2007 was the start since it is the first day of the availability of the Tor consensuses.

6.3.2 The threshold for the frequency is not handmade

A threshold had to be determined for the frequency. Indeed, the program has to decide if a BGP announcement by an AS not supposed to announce the prefix is abnormal or normal. BGP hijacks are most of the time one announcement that has never been done before. The frequency of this announcement must then be really low. The paper of counter RAPTOR [4] considers a BGP hijack below the threshold of 0.0025. As their results were positive, the threshold chosen was the one already taken by previous researchers.

6.3.3 The threshold for the time analysis is low, in order to avoid high false positive

As the time of the analysis is done over 13 years in theory (only three years of data were computed), be drowned into a huge quantity of potential BGP hijack was not a wanted scenario. Therefore, the threshold has been decreased to one hour. As expected, the number of potential hijacks detected by the time analysis is low but maybe too low.

6.3.4 The time analysis does not take smaller prefixes in account

When the origin check tells us that the prefix being announced is not from the supposed AS, this BGP announcement is stored with its timestamp inside a dictionary. If under one hour, the good origin AS announces this same prefix, it will write the malicious announcement as a potential BGP hijack. The true origin AS could also have announced a more specific prefix. In theory, this would also have been a defend against a BGP hijack. In this program, this scenario was not considered. The main reason is that the solution has not been found before launching the long computation.

6.4 Further improvements

A less ressource consuming approach would have been to use a a smaller number of BGP collectors. The results would have been less precise but we would had been able to compute the results over the full time period as initially planned.

Then, the thresholds used were not optimised and probable improvements could be reached by investigating different values. This is something complex as the issue of using a threshold is in fact to find a good one.

The program is nevertheless working and all the results could be computed in about one year.

CHAPTER 7

Result

In this chapter, the result of both programs will be shown and analyzed. We will then provide a discussion.

7.1 Result of resilience computation

The resilience of Tor relays on BGP hijack has been computed each 1st November from November 2007 to November 2015. It starts this month because the Tor consensuses are only available from the 27 October 2007. First, the result obtained will be illustrated. Then, the result will be analyzed. Finally, our results will be compared with the ones of previous research.

7.1.1 Global information of the computation

As explained above, this program uses a lot of resources. The high consumption of these is from the size of the virtual network and the number of prefixes announced inside it. Here are some graphs about these parameters for each year of computation. Figure 7.1 shows the variation in the number of prefixes for each year. Figure 7.2 shows the variation of the size of the network for each year.

The number of theses keeps increasing progressively. So a more recent computation of the resilience will take more time and resources than an oldest one.

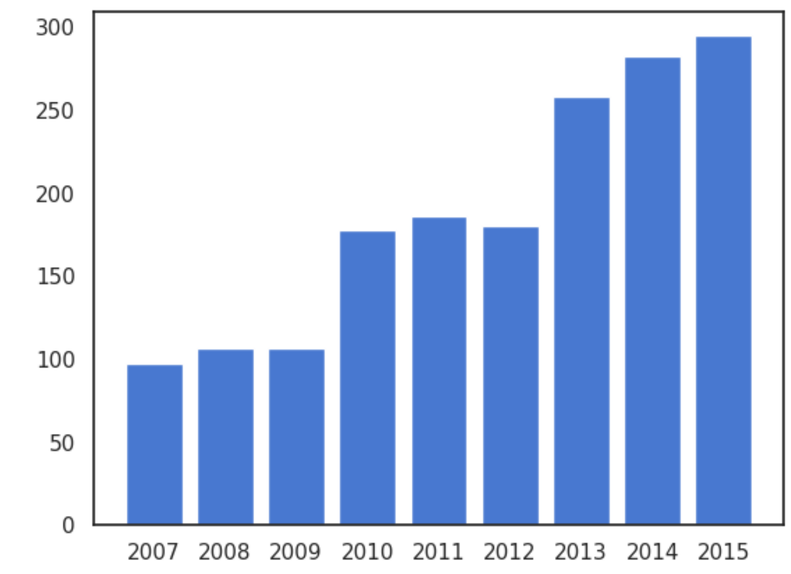


Figure 7.1: Number of prefix containing Tor relays extracted from the BGP archives

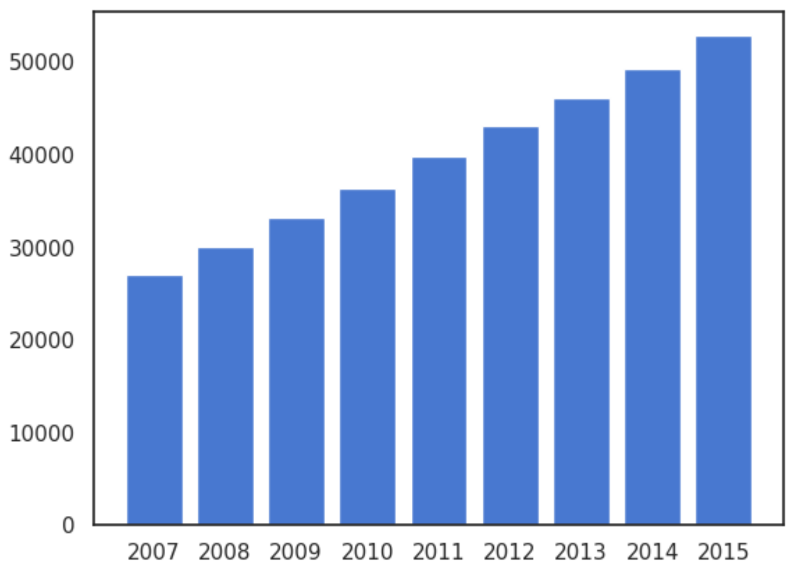


Figure 7.2: Number of AS inside the virtual network

7.1.2 Resilience obtained

We computed the resilience of Tor relay on BGP hijack each 1st November from 2007 to 2015. The distribution of the resilience computed is shown in figure 7.3 and the average of resilience depending on the year is shown in figure 7.4

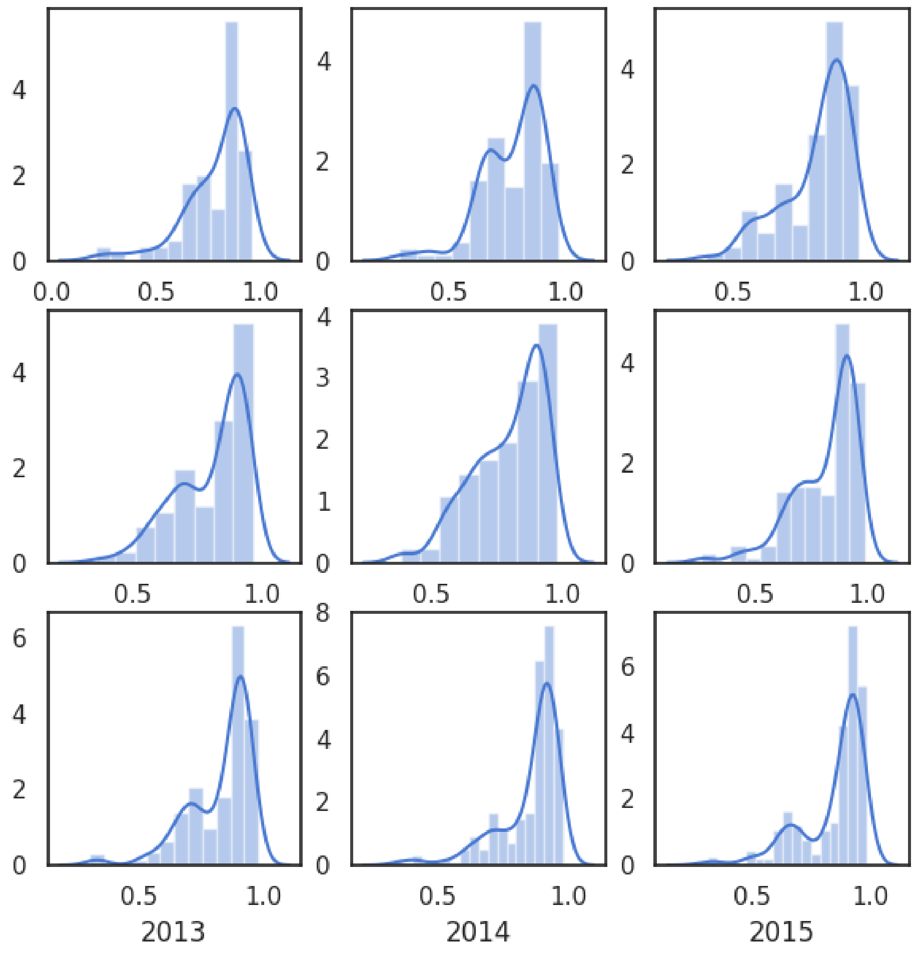


Figure 7.3: Distribution of the resilience computed by year

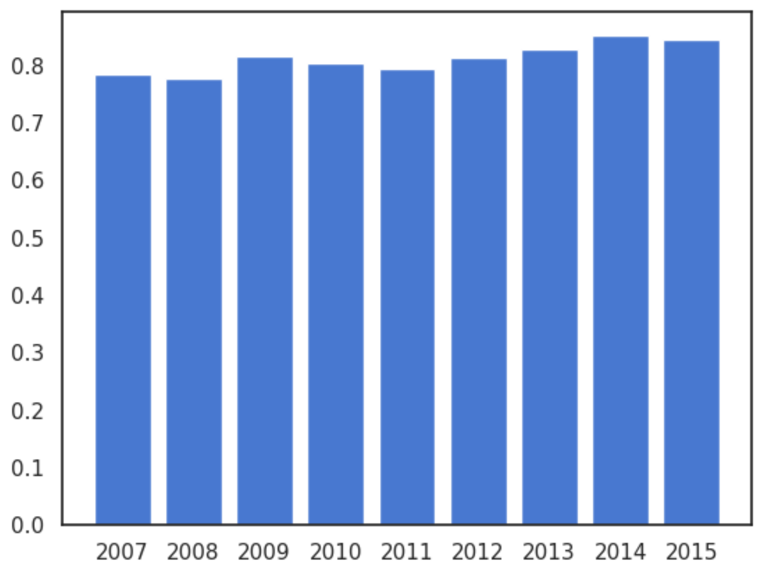


Figure 7.4: Average of the resilience computed by year

7.1.3 Analysis

We can see that the resilience are conservative from a year to another and exhibiting a bimodal behaviour with peaks around 0.6 and 0.8. However, the peak of 0.8 is much higher than the second one. This means that the majority of the resilience is at 0.8 and that the second majority of the resilience is at 0.6.

The average of resilience is nearly the same from one year to another. It tends to 0.8 of resilience.

7.1.4 Comparing with previous research

In [4], the researcher computed the resilience of the 1st of January 2016. They computed two resilience different than our:

1. The resilience of only the guard Tor relay with an average of 0.5.

2. The resilience of all the relay composing Tor with an average of 0.4.

These results are different than the one obtained with our program. This difference could be explained by several facts from our program:

1. The computation has been done with the guard and exit relays.

2. The error rate of the computation of resilience has been increased by decreasing the number of potential attackers. The number of these has been set to 100.
3. They computed the resilience of the 1st January 2016 which our program has not computed.

7.2 Result of BGP hijack monitoring

7.2.1 Potential BGP Hijacked discovered

The program was run for during two months. During this period of time, it has filtered 3 years of BGP announcements. As explained in section 3, an origin check first was performed. For each unexpected AS prefixes announced, a frequency and time analysis was performed.

As the number of potential BGP hijack discovered with an origin check was found to be high, only results from bad frequency/time analysis were considered. The differences between the frequency and time analysis are described in section 3.3.4.

As the number of potential BGP hijack found with the program is reasonable over a 3 year period, they are presented in the following tables.

Found with frequency analysis

Here is a chronological list of the potential BGP hijacks found with the frequency analysis.

Date	Prefix	AS	Frequency
2007-10-30-23-00-00	195.177.250.0/23	35492	0.000847457627118644
2008-01-19-17-00-00	194.116.182.0/23	3	0.002336448598130841
2008-01-20-07-00-00	194.116.182.0/23	3	0.0013333333333333333
2008-01-20-07-00-00	194.116.182.0/23	10	0.0006666666666666666
2008-01-20-10-00-00	194.116.182.0/23	3	0.0022675736961451248

Found with time analysis

Here is a chronological list of the potential BGP hijack founded with the time analysis.

Date	Prefix	True AS	Wrong AS	Frequency
2007-12-13-08	125.162.188.0/22	17974	7713	117 alert, 2 to 3593
2007-12-13-12	125.162.188.0/22	17974	7713	266.0
2007-12-13-15	125.162.188.0/22	17974	7713	110.0
2007-12-13-15	125.162.188.0/22	17974	7713	3011.0
2007-12-13-23	125.162.188.0/22	17974	7713	55.0
2007-12-13-23	125.162.188.0/22	17974	7713	38.0
2007-12-13-23	125.162.188.0/22	17974	7713	38.0
2007-12-13-23	125.162.188.0/22	17974	7713	995.0
2008-01-17-07	125.162.188.0/22	17974	7713	63 alert, 1 to 3394
2008-01-17-08	125.162.188.0/22	17974	7713	1987.0
2008-01-17-08	125.162.188.0/22	17974	7713	2006.0
2008-01-17-08	125.162.188.0/22	17974	7713	1363.0
2008-01-17-09	125.162.188.0/22	17974	7713	38.0
2008-01-17-09	125.162.188.0/22	17974	7713	42.0
2008-01-17-09	125.162.188.0/22	17974	7713	37.0
2008-01-17-09	125.162.188.0/22	17974	7713	39.0
2008-01-17-10	25.162.188.0/22	17974	7713	12 alert, 32 to 39
2009-04-11-21	8.103.167.0/24	46841	174	30.0
2009-04-11-21	8.103.167.0/24	46841	174	25.0
2009-04-11-21	8.103.167.0/24	46841	174	8.0
2009-04-11-22	38.103.167.0/24	46841	174	26.0
2009-04-11-23	38.103.167.0/24	46841	174	60.0
2009-04-11-23	38.103.167.0/24	46841	174	20.0
2009-04-12-00	38.103.167.0/24	46841	174	9.0
2009-04-12-00	38.103.167.0/24	46841	174	1.0
2009-04-12-01	38.103.167.0/24	46841	174	29.0
2009-05-11-12	212.42.236.0/24	13214	12732	13.0
2009-05-11-12	212.42.236.0/24	13214	12732	119.0

7.2.2 Analysis

The following tools were used for the investigations of the alert made by the program:

- IANA IPv4 address space [30]: A webpage showing the IPv4 address space given to the different Regional Internet Registry.
- AS Rank [31] : A website giving information about ASes.

- Ripe NCC database [32]: Information about history of prefixes announced by Ases inside this RIR.
- Arin database [33]: Information about history of prefixes announced by Ases inside this RIR.

Frequency analysis

The program detected two events that could be potential BGP hijacks. None of them were also detected by the time analysis.

1. 30 October 2007, the prefix 195.177.250.0/23 has been announced by AS 35492 instead of AS 1853. This prefix contains one IP address of a Tor exit relay.

   AS 1853 is owned by ACOnet. It is the national research and education network in Austria[34]. This AS is being managed by the University of Vienna and it seems coherent for them to run Tor relays.

   AS 35492 is also an AS from Austria. It is called FunkFeuer - Association for the promotion of free networks. It is a business AS[35]. Here is their description: " FunkFeuer is a free, experimental community network. Our mission is to provide free communication and access to information without commercial interests or providers, and we jointly operate independent infrastructure."

   This alert looks to be an error as the goal of FunkFeuer is to create a radio network linked to the Internet. Everyone can join and extend this network since 2003. I do not think that they were really interested in making correlation attacks on Tor network traffics in 2007.

2. The night of the 19 to the 20 January 2008, the prefix 194.116.182.0/23 as been announced by AS 3 and 10 instead of AS 42986. This prefix contains one IP address of a Tor exit relay.

   AS 42986 is owned by IZACOM. It was established in 1995 and it deals with providing Internet access to individual clients and companies in Poland[36].

   AS 3 and 10 are AS from the United States. AS 3 is owed by the Massachusetts Institute of Technology and AS 10 is owned by CSNET-EXT-AS. An old AS managing the Computer Science Network (CSNET) created at the begin of the Internet.

This event is even stranger. The MIT trying to hijack a prefix from a small AS lost in Poland. This might be an announcement made by the network researchers at the time for some experiments.

Time analysis

The program detected three events that could be potential BGP hijack. None of them were also detected by the frequency analysis.

1. 13 December 2007 to 17 January 2008, the prefix 125.162.188.0/22 (composed of one IP address of a Tor exit relay) has been announced several times by the AS 7713 instead of the AS 17974. After doing research on AS rank, I learned that AS 7713 and 17974 are from the same telecommunication group called "Telekomunikan Indonesia PT". This is not a BGP hijack.

   This event still continues. The 17 January 2008, the AS 17974 announces nearly the same prefix: 25.162.188.0/22. It is the same as before but without the 1 of the 125. The latter is owned by the UK Ministry of Defense. There are two possible scenarios:
   A. This is a misconfiguration from the network operator. He missed the number one in front of the prefix [10].
   B. This is a wanted scenario and they wanted to be stealthy by defending themselves with scenario A.

2. 11 April 2009, the prefix 8.103.167.0/24 (composed of one IP address of a Tor exit relay) has been announced several times by AS 46841 instead of "AS 174". This is what the program tells us but I suspect an error as AS 174 is not the right AS to announce this prefix. The prefix is owned by Arin (8.0.0.0/9).

   The next day nearly the same prefix is announced by AS 46841 : 38.103.167.0/24 which contains one Tor exit relay. It is the same with the 3 in front of the prefix. This prefix is owned this time by AS 174. I learned on AS rank that these AS are from the same telecommunication group called "Forknetworking".

   This is the same scenario as before but in the other way. Is it a miss-configuration? Is it intentional?

3. 11 May 2009, the prefix 212.42.236.0/24 (composed of one IP address of a Tor exit relay) is announced several times by AS 13214 instead of AS 12732. AS 13214 is owned by Hans Fredrik Lennart Neij[37]. He is the co-founder of The Pirate Bay which is an online index of digital content of entertainment media and software[38]. This AS is from Sweden. AS 12732 is owned by

GutCon GmbH. I did not find other information about it except that is it from Germany. This event is suspicious and might be a BGP hijack.

7.3 Discussion

The metrics proposed have drawbacks. We will use this section to discuss them.

Resilience

First, the resilience proposed can be reconsidered as a simpler computation than the one proposed in the state of the art has been done. Instead of simulating the situation where all the AS are potential attackers, only 100 attackers were chosen.

Then, during the propagation of the malicious announcement, the BGP process prefers the new path if it has the same level of preference as the old one. The real resilience computation proposed in the counter RAPTOR [4] would make a computation on these two paths. The proposed program will choose the new path and then only make the result based on one path.

Finally, due to performance issues, not all the resilience initially planned was computed. Performance issues could be twofold:

1. Our program is not enough optimized.
2. Our program is optimized but more resources are needed to make the computation.

Monitoring

Moreover, the majority of the potential BGP hijacks discovered does not seem to be BGP hijacks. The majority of them seems to be configuration errors made by the network operator, as the malicious announcements are often really close to the real prefix that they were allowed to announce.

The threshold proposed can be reconsidered as they do not really make alert about BGP hijacks whom could be attacks. A proposition could be to use bigger thresholds than the one used inside this program.

CHAPTER 8

Conclusion

Tor is a network that provides anonymity and security to its users from untrusted destinations and third parties on the Internet. It is one of the biggest anonymity networks deployed. Within the proposed metrics on the *Tor Metrics* website, none are about security. It would be valuable for the community if security-related metrics were proposed. In a correlation attack, if an attacker has access to traffic going in and out of the Tor network, he can correlate both traffic and the user loses his anonymity. To be able to see both sides of the traffic, the attacker can make BGP hijack to redirect traffic. Defenses exist for this type of attack, but none can definitively counter the attack at this time.

In this master thesis, new security metrics were proposed for the Tor network to tell which part of the network is more robust against BGP hijack, giving hence a visualization of security threat to the users.

Two metrics have been proposed. The first one is the resilience of Tor relay on BGP hijack each first day of the month. Unfortunately, due to optimisation issues, only nine days of data were analysed (1st Novemebr 2007 to 1st November 2015). Then, the second metric is a list of potential BGP hijacks. Three years of BGP archives have been filtered to catch these potential BGP hijacks.

The distribution of resilience exhibit a bimodal behaviour with peaks around 0.6 and 0.8 and an average around 0.8. This seems to be consistent from a year to another. Potential BGP attacks were discovered within a period of three years of data. Two events detected with the frequency analysis and three events detected with the time analysis were further analysed and discussed.

Further work

The programs have to be optimized to be able to compute the metrics from the beginning of the Tor network to current time. Moreover, other languages than Python could be investigated in order to see if performance could be improved.

Finally, the metrics could also be embellished. The Tor relay resilience and the potential BGP hijack could be merged in one metric. It could be shown in a graph. An idea would be to map all the Tor relays on a geographic map and to put the relay in different colors depending on their resilience and the history of the potential BGP hijacked. Green would be good robustness on BGP hijack and red a bad one.

Link to project repository

The code and resources can be found in the following Github repository:
https://github.com/Wvisee/RAPTOR.git

Bibliography

[1] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. Technical report, Naval Research Lab Washington DC, 2004.

[2] Tor metric. https://metrics.torproject.org/. Accessed: 2020-04-29.

[3] Yixin Sun, Anne Edmundson, Laurent Vanbever, Oscar Li, Jennifer Rexford, Mung Chiang, and Prateek Mittal. {RAPTOR}: Routing attacks on privacy in tor. In *24th {USENIX} Security Symposium ({USENIX} Security 15)*, pages 271–286, 2015.

[4] Yixin Sun, Anne Edmundson, Nick Feamster, Mung Chiang, and Prateek Mittal. Counter-raptor: Safeguarding tor against active routing attacks. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 977–992. IEEE, 2017.

[5] Tor Project overview. https://2019.www.torproject.org/about/overview.html.en. Accessed: 2020-04-24.

[6] Transmission Control Protocol wikipedia. https://en.wikipedia.org/wiki/Transmission _Control_Protocol. Accessed: 2020-04-24.

[7] Onion Routing wikipedia. https://en.wikipedia.org/wiki/Onion_routing. Accessed: 2020-04-24.

[8] BGP wikipedia. https://fr.wikipedia.org/wiki/Border_Gateway_Protocol. Accessed: 2020-04-24.

[9] UCL network_syllabus. https://beta.computer-networking.info/syllabus/default/index.html. Accessed: 2020-04-24.

[10] Shinyoung Cho, Romain Fontugne, Kenjiro Cho, Alberto Dainotti, and Phillipa Gill. Bgp hijacking classification. In *2019 Network Traffic Measurement and Analysis Conference (TMA)*, pages 25–32. IEEE, 2019.

[11] Vitaly Shmatikov and Ming-Hsiu Wang. Timing analysis in low-latency mix networks: Attacks and defenses. In *European Symposium on Research in Computer Security*, pages 18–33. Springer, 2006.

[12] Florentin Rochet and Olivier Pereira. Dropping on the edge: Flexibility and traffic confirmation in onion routing protocols. *Proceedings on Privacy Enhancing Technologies*, 2018(2):27–46, 2018.

[13] Benjamin Greschbach, Tobias Pulls, Laura M Roberts, Philipp Winter, and Nick Feamster. The effect of dns on tor's anonymity. *arXiv preprint arXiv:1609.08187*, 2016.

[14] Andriy Panchenko, Lukas Niessen, Andreas Zinnen, and Thomas Engel. Website fingerprinting in onion routing based anonymization networks. In *Proceedings of the 10th annual ACM workshop on Privacy in the electronic society*, pages 103–114, 2011.

[15] RPKI bgp-hijack. https://blog.cloudflare.com/rpki/. Accessed: 2020-04-25.

[16] RPKI bgp-hijack. https://medium.com/@nusenu/how-vulnerable-is-the-tor-network-to-bgp-hijacking-attacks-56d3b2ebfd92. Accessed: 2020-04-25.

[17] Jordan wright tor consensus explaination. https://jordan-wright.com/blog/2015/05/14/how-tor-works-part-three-the-consensus/. Accessed: 2020-07-28.

[18] Lixin Gao and Jennifer Rexford. Stable internet routing without global coordination. *IEEE/ACM Transactions on networking*, 9(6):681–692, 2001.

[19] Torproject tor network consensuses. https://collector.torproject.org/archive/relay-descriptors/consensuses/. Accessed: 2020-06-09.

[20] Ripe-NCC archives. https://www.ripe.net/analyse/internet-measurements/routing-information-service-ris/ris-raw-data. Accessed: 2020-06-08.

[21] Caida as-relation. http://data.caida.org/datasets/as-relationships/serial-1/. Accessed: 2020-07-27.

[22] mrt2bgpdump by t2mune. https://github.com/t2mune/mrtparse/blob/master/examples/mrt2bgpdump.py. Accessed: 2020-07-29.

[23] Ubuntu 18.04. https://releases.ubuntu.com/18.04/. Accessed: 2020-07-30.

[24] Oom killer. https://docs.memset.com/other/linux-s-oom-process-killer. Accessed: 2020-07-30.

[25] ipv6 bgp length. https://labs.ripe.net/Members/stephen_strowes/visibility-of-prefix-lengths-in-ipv4-and-ipv6. Accessed: 2020-07-30.

[26] Team Cymru ip to asn mapping. https://team-cymru.com/community-services/ip-asn-mapping/. Accessed: 2020-06-08.

[27] Python3 copy. https://docs.python.org/3/library/copy.html. Accessed: 2020-08-07.

[28] Python3 pickle. https://docs.python.org/3/library/pickle.html. Accessed: 2020-08-11.

[29] Routeviews archives. http://archive.routeviews.org. Accessed: 2020-06-08.

[30] IANA IPV4 space. https://www.iana.org/assignments/ipv4-address-space/ipv4-address-space.xhtml. Accessed: 2020-08-03.

[31] ASrank caida. https://asrank.caida.org/. Accessed: 2020-08-03.

[32] Ripe NCC db. https://apps.db.ripe.net/db-web-ui/query. Accessed: 2020-08-03.

[33] Arin database. https://search.arin.net/rdap/. Accessed: 2020-08-03.

[34] ACOnet. https://en.wikipedia.org/wiki/ACOnet. Accessed: 2020-08-03.

[35] Website funkfeuer. https://funkfeuer.at/. Accessed: 2020-08-03.

[36] Website izacom. http://home.izacom.pl/. Accessed: 2020-08-03.

[37] Fredrik neij. https://en.wikipedia.org/wiki/Fredrik_Neij. Accessed: 2020-08-03.

[38] Pirate bay. https://en.wikipedia.org/wiki/The_Pirate_Bay. Accessed: 2020-08-03.