FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of bachelor's thesis

Title:	Security analysis of Globalping probes
Student:	Viktor Tyúkos
Supervisor:	Ing. Martin Kolárik
Study program:	Informatics
Branch / specialization:	Information Security 2021
Department:	Department of Information Security
Validity:	until the end of summer semester 2024/2025

Instructions

Globalping is an open-source platform that allows anyone to run networking commands (ping, traceroute, dig, curl, and mtr) on probes distributed around the world. The probes are run mainly by volunteers, often in their homes or other private networks.

Proceed in the following steps:
1. Familiarize yourself with the Globalping platform features, architecture, and security practices.
2. Research selected competing platforms and compare them in terms of features and security practices. Discuss security-sensitive areas of the probes, possible attacks, and existing countermeasures.
3. Analyze the source code of a Globalping probe and discuss your findings in terms of the probe's security.
4. If any security issue is found, include a risk assessment and a recommendation for resolving it.

Electronically approved by prof. Ing. Róbert Lórencz, CSc. on 18 December 2023 in Prague.

Bachelor's thesis

SECURITY ANALYSIS OF GLOBALPING PROBES

Viktor Tyúkos

Faculty of Information Technology
Department of Information Security
Supervisor: Ing. Martin Kolárik
May 16, 2024

Citation of this thesis: Tyúkos Viktor. *Security analysis of Globalping probes.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2024.

Contents

List of Figures

List of Tables

List of code listings

First, I would like to thank my supervisor, Ing. Martin Kolárik, for providing me with his knowledge and expertise while writing this thesis. Additionally, I would like to express my gratitude to my family and friends for all the support that I have received during my studies.

Declaration

Abstract

This thesis covers the topic of security analysis of the Globalping probe. It begins by introducing the Globalping platform and comparing it to some of its competitors. The thesis then proceeds to model the implementation of the probe itself and the threat environment in which the Globalping probe operates. Following that, the focus shifts to analyzing the security features of the probe and its implementation. This process involves validating all of the claims made by the Globalping development team and conducting a security analysis of the probe source code. Although some security issues were discovered, none of them are critical but there remains a space for improvement. Finally, all of the findings are summarized, and recommendations for their mitigation are provided.

Keywords network probe, source code analysis, Globalping, jsDelivr, vulnerability analysis, threat modeling

Abstrakt

Tato práce se zabývá tématem bezpečnostní analýzy sondy Globalping. Začíná představením platformy Globalping a jejím porovnáním s některými jejími konkurenty. Poté práce pokračuje modelováním implementace samotné sondy a prostředí hrozeb, ve kterém sonda Globalping pracuje. Poté se pozornost přesouvá na analýzu bezpečnostních prvků sondy a její implementace. Tento proces zahrnuje ověření všech tvrzení vývojového týmu Globalping a provedení bezpečnostní analýzy zdrojového kódu sondy. Přestože byly objeveny některé bezpečnostní problémy, žádný z nich není kritický, ale zůstává prostor pro zlepšení. Na závěr jsou shrnuta všechna zjištění a uvedena doporučení pro jejich zmírnění.

Klíčová slova síťová sonda, analýza zdrojového kódu, Globalping, jsDelivr, analýza zranitelnosti, modelování hrozeb

List of Acronyms

API	Application Programming Interface
CDN	Content Delivery Network
CVSS	Common Vulnerability Scoring System
DDOS	Distributed Denial of Service
DFD	Data Flow Diagram
DNS	Domain Name Service
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IDE	Integrated Development Environment
IP	Internet Protocol
NAT	Network Address Translation
NTP	Network Time Protocol
SSL	Secure Socket Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol

Introduction

Globalping is an open-source platform that allows anyone to run networking commands like ping, traceroute, dig, curl, and mtr on probes distributed around the world. These probes are mainly run by volunteers, often within their homes or other private networks. For that reason, for this network of probes to exist, the probes themselves need to be secure and trustworthy, to gain the faith of the volunteers. Furthermore, the security of these probes is important to prevent the disclosure of sensitive information or to prevent the abuse of this network as a botnet. Therefore, the primary objective of this thesis is to perform a security analysis on these probes, ensuring their safety and that they do not introduce new attack vectors on the machines or network of the volunteers, who run them. As such this thesis will mainly be of interest to people that are interested in hosting their probes and contributing to this global network.

Goals

The goal of this thesis is to perform a security analysis of the Globalping probes. In particular, this thesis will consist of:

- Analyzing the Globalping platform,
- Exploring possible competitors and comparing them in terms of security and features,
- Performing a threat model and source code analysis on the Globalping probes,
- Suggesting remediation to any vulnerabilities discovered during the source code analysis.

My personal goals consist of:

- Get hands-on experience with security analysis,
- Broaden my horizons when it comes to the security of network probes,
- Build upon the knowledge of computer networks that I acquired during my studies.

Chapter 1

The Globalping platform

This chapter provides information about the features, architecture, and security practices of the Globalping platform.

1.1 The Globalping platform

Globalping is a platform created by jsDelivr that enables users to execute networking commands such as `ping`[1], `traceroute`[2], `dig`[3] , `curl`[4], or `mtr`[5] on probes distributed all over the world through a RESTful API (Application Programming Interface). This platform can be leveraged to optimize your anycast network, monitor latency, debug routing, or even check for censorship across different countries.

This is being accomplished through a series of probes, which can be hosted by anyone from CDN (Content Delivery Network) providers to individual people. Access to this network of probes can be achieved either through downloading a command-line application, usage of the Globalping website or even taking advantage of Slack integration with their Slack app.

As for running a probe, it is as simple as downloading and running a docker container on a machine connected to the internet or obtaining one of the hardware probes by donating to the Globalping project.

To use the Globalping platform to perform tests, the users do not even need to be registered. As such, anyone can run tests using Globalping. For a better understanding of these tests, you can reference the sample request in Listing 1.1 and the sample reply in Figure1.1 on the Globalping platform and the list of available tests below. [1]

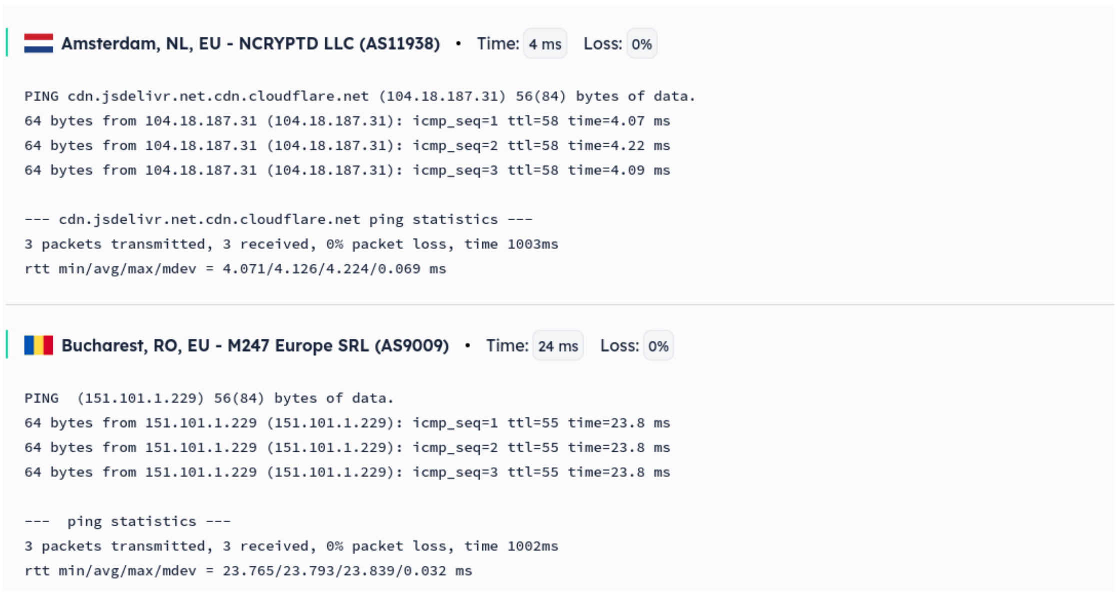
- **Ping**, allows for the measurement of network connectivity and latency between the probes and the target.
- **Traceroute** is used for displaying the network route and latency of each hop on the network.
- **DNS (Domain Name Service)**, using the `dig` command to query DNS records of the specified domain.
- **HTTP (Hypertext Transfer Protocol)**, combines the `curl` GET and HEAD utilities and the output is limited to 10kb.

[1] https://linux.die.net/man/8/ping
[2] https://linux.die.net/man/8/traceroute
[3] https://downloads.isc.org/isc/bind9/9.18.26/doc/arm/html/manpages.html#dig-dns-lookup-utility
[4] https://curl.se/
[5] https://linux.die.net/man/8/mtr

- **mtr**, combines the utilities of the `ping` and `traceroute` commands.

```
{
  "type":"ping",
  "inProgressUpdates":true,
  "target":"cdn.jsdelivr.net",
  "limit":"10",
  "locations":[
    {
      "magic":"World"
    }
  ],
  "measurementOptions":{
    "packets":3
  }
}
```

Listing 1.1 Sample request to the Globalping API to perform a measurement.



Amsterdam, NL, EU - NCRYPTD LLC (AS11938) • Time: 4 ms Loss: 0%

```
PING cdn.jsdelivr.net.cdn.cloudflare.net (104.18.187.31) 56(84) bytes of data.
64 bytes from 104.18.187.31 (104.18.187.31): icmp_seq=1 ttl=58 time=4.07 ms
64 bytes from 104.18.187.31 (104.18.187.31): icmp_seq=2 ttl=58 time=4.22 ms
64 bytes from 104.18.187.31 (104.18.187.31): icmp_seq=3 ttl=58 time=4.09 ms

--- cdn.jsdelivr.net.cdn.cloudflare.net ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1003ms
rtt min/avg/max/mdev = 4.071/4.126/4.224/0.069 ms
```

Bucharest, RO, EU - M247 Europe SRL (AS9009) • Time: 24 ms Loss: 0%

```
PING  (151.101.1.229) 56(84) bytes of data.
64 bytes from 151.101.1.229 (151.101.1.229): icmp_seq=1 ttl=55 time=23.8 ms
64 bytes from 151.101.1.229 (151.101.1.229): icmp_seq=2 ttl=55 time=23.8 ms
64 bytes from 151.101.1.229 (151.101.1.229): icmp_seq=3 ttl=55 time=23.8 ms

---  ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1002ms
rtt min/avg/max/mdev = 23.765/23.793/23.839/0.032 ms
```

Figure 1.1 Reply to the sample request to perform a measurement.

1.2 Architecture

The Globalping virtual probe is an open-source application written in TypeScript that has been packaged in a Docker container, which allows it to run on any platform. When a user requests to execute a network test, the Globalping API accepts the request and forwards it to the relevant probe to be carried out. The probe then performs the test and replies to the API with the results, which then replies to the user. For a more detailed look, you can refer to the data flow diagram 1.2 below. [2]

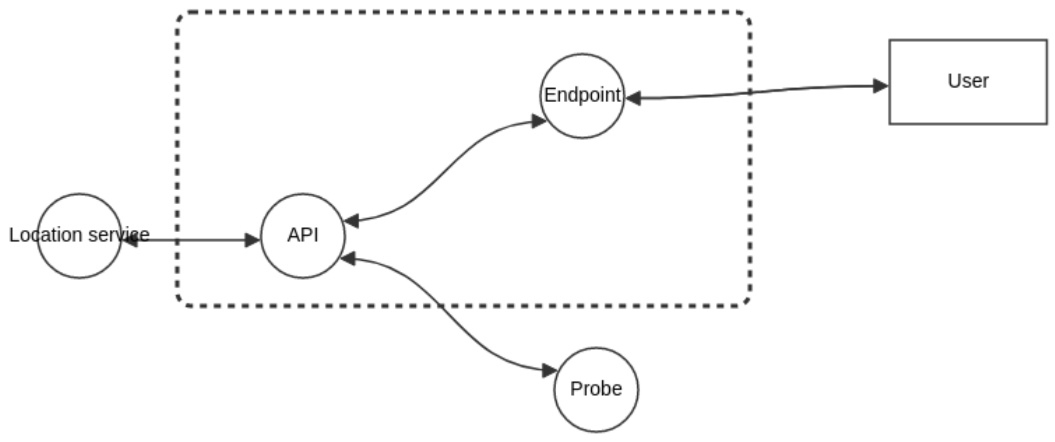


Figure 1.2 Architecture of the Globalping Platform

1.3 Security practices

When it comes to security, the Globalping platform implements several security measures to protect its users from any potential threats. The platform limits its users to hosting a single probe per public IP (Internet Protocol) address, whether it is a software or hardware probe. Additionally, the probe does not use a public IP address, making it possible to be placed inside a private network behind NAT (Network Address Translation). Furthermore, the probe does not open any TCP/UDP ports on the host machine.

All communication between the probe and the API is carried out through the HTTPS (Hypertext Transfer Protocol Secure) protocol using TLS 1.3 (Transport Layer Security). Ensuring, that the communication cannot be intercepted.

The Globalping platform limits its users to 100 POST requests to the API per minute per IP address and a single measurement is limited to 200 probes in a single location or 500 probes worldwide. Unauthenticated users can perform 100 tests per hour, while authenticated users have this limit raised to 200 tests. A test is defined as a successful measurement, which means that 100 tests can be understood as measuring 100 targets with a single probe or using 100 probes to measure a single target. This limitation helps prevent DOS (Denial of Service) attacks on the probe or the platform as a whole. Additionally, it helps in preventing the flooding of the user networks.

The Globalping platform keeps a database of domains and IP addresses associated with malware or malicious activities and actively blocks them. The platform forbids and blocks the measurement of private IP addresses, only public endpoints are valid targets for the tests. [3]

Chapter 2

Competing platforms

The following chapter will discuss the main competitors of the Globalping platform and their approaches. Several platforms provide services that are similar to Globalping, some examples may include DNSPerf, CDNPerf, PerfOps, and RIPE Atlas.

Most of these cannot be compared to Globalping as they either lack the features of Globalping, do not offer an open network of probes, or both. As such it was decided to compare the Globalping platform with the RIPE Atlas platform as it offers similar testing capabilities and uses an open network of probes, at least partially hosted by users.

2.1 RIPE Atlas

RIPE Atlas, founded by the RIPE NCC (Network Coordination Centre), is a global network of over 12000 probes. This platform offers capabilities to measure Internet connectivity and reachability in real-time, providing insights into the state of the Internet. The data collected by the platform is freely available, making the platform a popular choice with researchers.

While it is possible to create custom measurements, this feature requires credits that can be earned either by hosting a probe or purchasing them. However, the process of hosting a probe is not as straightforward as in the case of Globalping. If a user wants to host a RIPE Atlas probe, first it is required to send a request to RIPE, which will then evaluate the need for a new probe in the user's location before deciding whether to send a probe to the new host. [4]

2.1.1 Features

The network of probes is used by RIPE Atlas to conduct the following types of measurements:

- **Ping**, allows for the measurement of network connectivity and latency between the probes and the target.
- **Traceroute** is used for displaying the network route and latency of each hop on the network.
- **DNS (Domain Name Service)**, using the `dig` command to query DNS records of the specified domain.
- **SSL (Secure Socket Layer)/TLS**, allows displaying of the TLS certificate used by the target.
- **NTP (Network Time Protocol)**, allows for the measurement of the time difference between a target and the probes.

- **HTTP (Hypertext Transfer Protocol)**, allows for sending `curl` HEAD requests and measuring the latency of the connection to the target.

RIPE Atlas provides two types of measurements: built-in and custom (user-defined).

Built-in measurements consist of `ping`, `traceroute`, DNS, SSL/TLS, and limited HTTP. These measurements are automatically performed by all probes at regular intervals, using root DNS servers and also some parts of the RIPE Atlas infrastructure as their targets. The results of these measurements are publicly available on the RIPE Atlas website.

User-defined measurements can be carried out by RIPE Atlas probe hosts, anchor hosts, sponsors, and NCC members. These custom measurements allow users to perform network tests tailored to their individual needs. To create a custom measurement, the user is required to obtain credits through hosting a probe, being a sponsor, being a RIPE NCC member, purchasing, or through a transfer from another user.

A RIPE Atlas anchor acts both as an enhanced probe with higher measuring capacity, as well as a regional target within the global RIPE Atlas network, making them able to collect information from a number of RIPE Atlas probes. [5]

2.1.2 Feature Comparison

Both platforms offer their unique benefits. While Globalping by JsDelivr is geared more towards the average user with its free real-time measurements, more streamlined UX, and a more modern open-source approach, RIPE Atlas offers a more detailed and regular network analysis with publicly available internet health data, which could be more beneficial to researchers and network administrators. A more detailed comparison of features can be found in Table 2.1 below.

[6]

2.1.3 Security practices comparison

RIPE Atlas and Globalping share several security measures. These measures include avoiding the opening of new TCP/UDP ports, supporting only outbound connections, and prohibiting measurements of private networks.

In addition to these measures, RIPE Atlas uses mutual authentication between hosts and infrastructure components. This makes obtaining a probe more difficult and helps prevent attackers from connecting to the RIPE Atlas network with a spoofed probe.

While RIPE Atlas uses a more precise probe location, the exact location is obfuscated in a radius of 1km to maintain user privacy.

The RIPE Atlas platform is built using shell, which may not be as secure as TypeScript. However, the platform conducts periodic security tests on its infrastructure and publishes the findings on its website. This demonstrates an initiative to keep the probe secure and up-to-date with modern security standards. [7]

Platform	**Globalping**	**RIPE Atlas**
Probe Network	Quickly growing network, where anyone can host a probe	Extensive global network. You need to apply for the opportunity to host a probe
Measurement types	Real-time focused measurements of: `ping`, `traceroute`, DNS, HTTP (including SSL), and `mtr`.	Scheduled measurements of `ping`, `traceroute`, DNS, HTTP, SSL, and NTP.
Measurement and Costs	Free of charge, but rate limited, the limit can be raised by becoming a sponsor	Measurements locked behind credits, that need to be obtained
Internet Health Monitoring	Does not offer public internet health monitoring	Offers public access to all measurements
Custom monitoring capabilities	yes	yes
Use cases	Designed to be used by users of various backgrounds	Tailored for use by researchers, academics or anyone who need aggregated internet connectivity data
Commercial use	Does not differentiate between private and commercial use	You need to follow certain rules for commercial use
Data usage	Measurement results are stored only temporarily and only users with the measurement link can view them	All measurements are publicly available
Tools and Integration	Provides user-friendly tools and integration for custom development	It is possible to find open-source tools
Getting Involved	Encourages users to host probes or to contribute on GitHub	Encourages you to host probes or anchors and to create open-source tools

Table 2.1 Comparison of Globalping and RIPE Atlas feature set

Chapter 3

Used methods

This section will cover the methodologies and general approach taken during the security analysis of the Globalping platform.

The whole process can be divided into two parts. First, it is necessary to create a threat model of the Globalping probe to assess possible threats and areas of focus for the analysis. Then, the analysis itself will be performed. Both of these parts will require the usage of several methodologies when it comes to achieving their respective goals and then evaluating and scoring the results of these processes.

The following sections will provide further insight into both of these parts.

3.1 Threat modeling

Threat modeling is a process used to identify, understand, communicate, and mitigate threats within a given context. A threat model is a structured representation of these findings, which can be used to give a view into the environment through the lens of security. A typical threat model includes:

- Description of the subject to be modeled.
- Assumptions that can be checked or challenged in the future as the threat landscape changes.
- Potential threats to the system.
- Actions that can be taken to mitigate each threat.
- A way of validating the model, threats, and the verification of the actions taken to mitigate these threats.

By combining all of this information it is possible to create a prioritized list of security improvements to the concept, requirements, and implementation of the application. Threat modeling should be a continuous process throughout the whole life cycle of the application. It should begin during the initial planning phase and continue through the development and implementation stages. [8]

In the context of security testing, threat modeling helps in focusing resources on the areas of the system that are under a threat of being exploited. Thus, making the process of identifying vulnerabilities more effective.

3.1.1 Threat categorization

It is important to not only identify system threats but also to categorize and rank them. We utilize the STRIDE and DREAD methodologies for categorization and ranking, respectively.

3.1.1.1 STRIDE

STRIDE is a threat modeling methodology used for categorizing threats into the following six categories:

- Spoofing,
- Tampering,
- Repudiation,
- Information Disclosure,
- Denial of Service,
- Elevation of Privilege.

The goal of STRIDE is to provide each threat with a category, that can be used to create a scenario of possible exploitation. For a more detailed description of these categories, please refer to table 3.1

Threat type	**Description**
Spoofing	Using stolen or forged credentials to access a restricted resource.
Tampering	Changing data in order to breach system security.
Repudiation	Occurs when there is no proof of actions being performed.
Information Disclosure	Disclosure of information to users, that do not have the authorization to see it.
Denial of Service	Is a type of attack that threatens the ability of a valid user to access a resource.
Elevation of Privilege	Is an attack where an unprivileged user gains privileged status.

Table 3.1 Breakdown of STRIDE categories

[9]

3.1.1.2 DREAD

Determining the potential occurrence and location of a threat to a tested application is not sufficient. It is also crucial to evaluate the severity of the identified threats. DREAD is an acronym that outlines the following criteria.

- Damage potential
- Reproducibility
- Exploitability
- Affected users
- Discoverability

To calculate the score for a particular threat, we assign a numerical value from 1 to 10 to each factor based on its severity level, with 1 being the lowest and 10 being the highest. We then add up the values assigned to these criteria and divide the resulting sum by 5. The higher the score, the more critical the threat. The DREAD methodology is divided into categories, which are explained in detail in Table 3.2. [9]

Threat type	**Description**
Damage potential	Assessing the damage resulting from an attack.
Reproducibility	A measure of how successful an attacker will be in reproducing the attack.
Exploitability	Describes the ease of the attack and the required skill level of the attacker.
Affected users	Describes the scale of the user base, that is going to be affected by this attack.
Discoverability	Describes how easy it is for the threat to be identified by an attacker.

Table 3.2 Breakdown of DREAD categories

3.2 Analysis

Source code analysis, also known as static code analysis, is usually carried out early in the implementation phase of the secure software development life cycle. It is performed as a part of white box testing. Source code analysis commonly refers to running automated tools that attempt to identify potential security vulnerabilities in a static, not running, source code. Many of these tools are now integrated into programming IDEs (Integrated Development Environment). However, it is important to note that these tools can not give definitive answers and there still exists a need to manually analyze the source code of the application. [10]

3.2.1 Used methodologies

There are various methods used to analyze source code. Some examples may include Data flow analysis, Taint analysis, and Lexical analysis. This work will utilize taint analysis to perform source code analysis and CVSS 3.1 (Common Vulnerability Scoring System Version 3.1) to assign scores to any vulnerabilities discovered during the analysis.

3.2.1.1 Taint analysis

In taint analysis, we attempt to identify variables that have been influenced by user input (‘tainted’) and trace them to vulnerable functions, also known as ’sinks’. If the tainted variable gets passed to a sink without being sanitized or validated, it is flagged as a vulnerability. [10]

3.2.1.2 CVSS 3.1

The Common Vulnerability Scoring System 3.1 (CVSS 3.1) is a qualitative measure of severity. It consists of three metric groups: base, temporal, and environmental. The base metric results in a score from 0 to 10, which can then be further modified by the temporal and environmental metrics. CVSS is an industry standard in scoring and ranking vulnerabilities, which is why it is the preferred methodology for this thesis. [11] The base metric consists of the following metrics:

- Attack Vector

- Attack Complexity
- Privileges Required
- User Interaction
- Scope
- Confidentiality
- Integrity
- Availability

Chapter 4

Threat Modeling the Globalping platform

This section will contain the threat modeling of the Globalping probe and discuss possible threats, their classification, and mitigation.

4.1 Scope

The purpose of this threat model is to get familiar with the Globalping probe application and how it interacts with other entities. Based on this collected information, I will assess the security threats to the probe, the host system, and the network of the host system. For this purpose, I will create a data flow diagram to help visualize the inner workings of the probe. Utilizing this new information will help in discovering and categorizing possible threats to the probe.

4.2 Decomposing the application

- **Application name:** Globalping probe
- **Application version:** 0.29.0[1],
- **Commit hash:** bcf95e75927b89834fc17af982665990de67a112,
- **Description:** The Globalping platform allows users to run a variety of networking commands on probes distributed around the world. Users can choose to host their own probe and contribute to this network. This analysis will further explore potential threats to these probes.

4.2.1 External dependencies

External dependencies are libraries outside the control of the development team. As such, from a security standpoint, it is important to monitor them for potential new vulnerabilities that then may be introduced into the application through their usage. Hence, it is paramount to keep these dependencies up-to-date.

Table 4.1 below contains the complete list of the external dependencies for the Globalping probe, that were discovered during the analysis of the source code.

[1] https://github.com/jsdelivr/globalping-probe/tree/v0.29.0

ID	Name	Version	Description
1	Docker		The probes run in a Docker container on the host machines.
2	Websocket		The communication with the API is done through Websockets.
3	execa	7.2.0	Execa runs commands in applications.
4	got	12.6.1	An HTTP request library.
5	http2-wrapper	2.2.0	Use http2 as http1
6	joi	17.11.0	A schema description language and data validator.
7	lodash	4.17.21	A JavaScript utility library delivering modularity.
8	physical-cpu-count	2.0.0	Physical CPU core detection
9	socket.io-client	4.7.2	Realtime application framework
10	throng	5.0.0	A multiplexor written in Java
11	tldts	6.0.15	A library to extract hostnames, domains, public suffixes, top-level domains, and subdomains from URLs.
12	winston	3.10.0	A simple and universal logging library.

Table 4.1 Table of external dependencies

4.2.2 Entry and exit points

Entry and exit points refer to the locations, where an application either receives or sends data. In the case of the Globalping probe, all of the communication occurs with the Globalping API. This communication flows through Websocket and all of the data supplied to the probe is the user input. This can lead to potential injection attacks. Therefore, it is important to take additional care during the analysis, especially in the area of sanitation and validation. 4.2

ID	Name	Description
1	Socket	The node listens for requests to perform tests. These requests arrive from the Globalping API.

Table 4.2 Table of entry points

4.2.3 Assets

Some assets of interest for the potential attacker might be the history of scans, but that is stored on the API side and is out of the scope of our analysis. That leaves us with the computing resources of the machine, that the probe is being run on and the network that it is connected to as the main points of interest for the attacker. The complete list with more detailed descriptions can be found in the Table 4.3 below:

ID	Description
1	Host system
2	Host data
3	Host network

Table 4.3 Table of assets

4.2.4 Trust levels

During our analysis, we differentiate between 2 trust levels. The first one is assigned to processes that are a part of the Globalping platform and the second one is assigned to external services, that are in use by the platform, but are less trusted, as they are not under the control of the Globalping development team.

ID	Name	Description
1	Trusted	Components under the control of the development team at Globalping.
2	Untrusted	External components. that are not under the control of the Globalping team.

Table 4.4 Table of assets

4.3 Data flow diagram

Based on the information received and analysis of the source code, a data flow diagram (DFD)4.1 has been compiled. The DFD represents the flow of data in the Globalping platform, with a focus on the probe part of the platform. The circles on the diagram represent logical processes running on the platform, whereas the arrows represent the flow of data.

Using this diagram and the information gathered, we have created a list of potential threats that will be useful in the next chapter. Please note that the API is not included in this analysis, but has been included in the diagram for the sake of clarity.

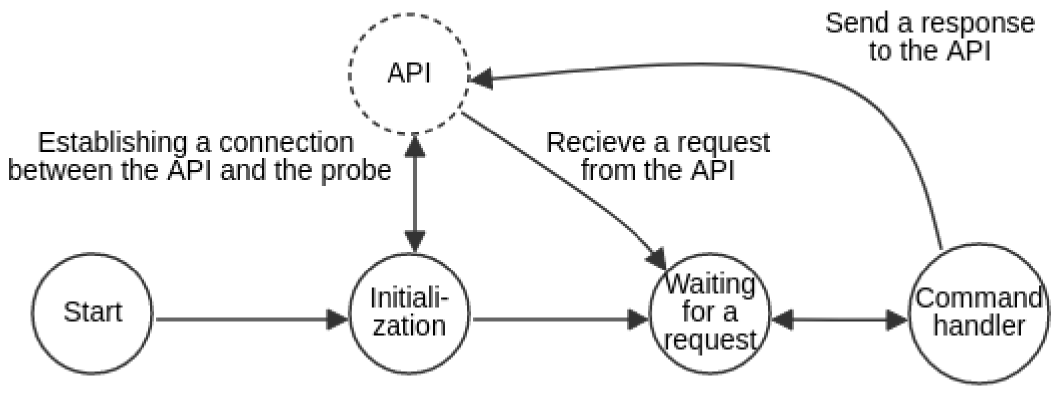

Send a response to the API
API
Establishing a connection between the API and the probe
Recieve a request from the API
Start
Initiali-zation
Waiting for a request
Command handler

Figure 4.1 Data flow diagram of the Globalping platform

4.4 Summary of threats

This section will provide the summary and classification of threats. These threats are based on the information obtained from the threat modeling process and require further validation. Each threat contains a short description and is classified using the STRIDE methodology. Additionally, each threat is rated according to the DREAD metric.

4.4.1 Injection

A component is vulnerable to injection attacks when user input is incorrectly interpreted and executed as code, instead of being handled correctly as data.

In the case of the platform, user input is forwarded to the probe to be executed as the payload of a shell function. As a result, attackers could exploit this vulnerability to try and execute their commands on the machines hosting the probes.

Classification
Classifying injection under the STRIDE methodology is a bit difficult, as it does not really fit any of the categories, or depending on the viewpoint, fits multiple. For this thesis, I have opted to go with the following categorization.

- **STRIDE:** Tampering
- **DREAD:** Injection vulnerabilities have the potential to cause a lot of damage to the application or its host. These types of attacks are not always straightforward and often require a certain level of skill to craft the right payload, but on the other hand, they can be easily replicated. As all of the users host the same probe, the whole network would be affected in case of an injection vulnerability. That gives a score of 7.6 with a more detailed breakdown below.
  - Damage potential: 9
  - Reproducibility: 8
  - Exploitability: 6
  - Affected users: 9
  - Discoverability: 6

4.4.2 Spoofing the probe

Spoofing occurs when a process or entity is disguised as something other than its claimed identity in order to deceive another party.

In the context of the Globalping platform, insufficient authentication of the connected probes could lead to an attacker connecting a program to the Globalping network, causing the platform to return incorrect values to supplied queries, collection of data by the attacker or other malicious activities.

Classification Classifying this threat is quite self-explanatory, which leads to the following classification:

- **STRIDE:** Spoofing
- **DREAD:** Spoofing the probe gives the attacker a reduced potential for damage. As the Globalping probe is an open-source application, the potential attacker does not need a lot of effort to change the source code, to alter the functionality of the probe. All of this combined produces the following score of 6.8.
  - Damage potential: 6
  - Reproducibility: 8
  - Exploitability: 8
  - Affected users: 6
  - Discoverability: 6

4.4.3 Denial of service

A component is vulnerable to Denial of service attacks when it can not serve legitimate incoming requests due to external factors.

In the case of the probe, it means that the probe is not able to perform measurements requested by the API, or even worse the probe can paralyze the API itself.

Classification

Denial of Service has its own category in STRIDE, which makes classifying it easy.

- **STRIDE:** Denial of service
- **DREAD:** Denial of service is not a difficult type of attack to be performed and can be easily reproduced. Nevertheless, it does not have a significant impact on the users hosting the probes. This gives us a score of 5.4.
  - Damage potential: 4
  - Reproducibility: 6
  - Exploitability: 7
  - Affected users: 4
  - Discoverability: 6

4.4.4 Host system information disclosure

Information disclosure occurs when a component reveals more information than the user is authorized to access.

As the probes are hosted on user machines, it is important to ensure that the probe does not disclose any information about the host machine.

Classification

Again, information disclosure has its own category under the STRIDE methodology, which makes classifying this threat straightforward.

- **STRIDE:** Information disclosure
- **DREAD:** Probes leaking host information would affect the whole Globalping network. Based on the associated infrastructure of the attack, this could prove to be trivial or quite difficult to exploit. As such it is scored as 7.
  - Damage potential: 6
  - Reproducibility: 6
  - Exploitability: 8
  - Affected users: 9
  - Discoverability: 6

Chapter 5

Analyzing the Globalping probe

Using the threats identified in the previous chapter, let us focus our attention on the in-depth source code analysis of the Globalping probe application.

5.1 Scope

The main focus of this analysis is based on the findings from the threat modeling, where the main focus will be on checking for various injection vulnerabilities in the codebase of the Globalping probe application. These vulnerabilities pose the greatest threat to the application and it is imperative. That the user-supplied input is correctly sanitized and validated. Naturally, the scope will not be limited only to these vulnerabilities, and tests for other types of attack will also be carried out. The tested version of the probe will be 0.29.0 with the particular git commit being: bcf95e75927b89834fc17af982665990de67a112.

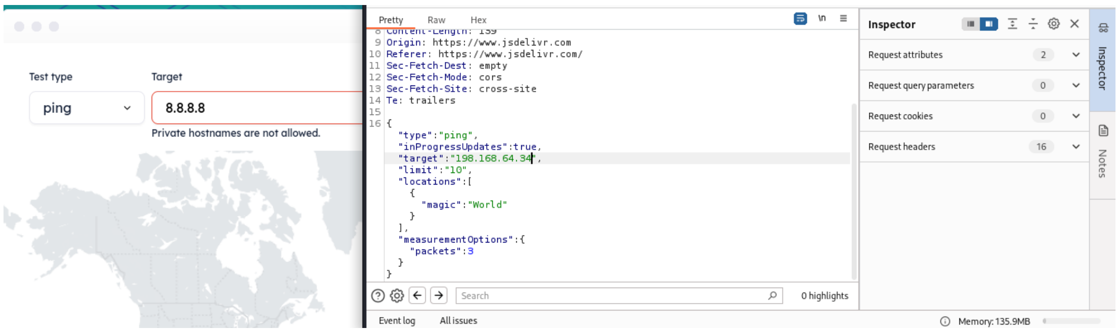
5.2 Validating the stated security claims

Let us begin by verifying the security features of the Globalping probe. Some of these features can be considered to be out of scope as they involve the API, but after consultation, it was decided to include them in this analysis.

- **Private IP detection**
  As can be seen in the provided example 5.1, the API correctly filters private IP addresses and this validation is not being done on the browser as it could not be circumvented by altering the request using BurpSuite. This feature also correctly detects domains, that have a DNS A record towards a private IP address.

- **Single probe per Public IP address**
  As can be seen in Figure 5.2, when trying to run 2 probes on a Single machine or using a separate virtual machine, the probe correctly detects another running instance and kills one of the instances.

- **No public IP address for the probe**
  From the packet capture 5.3, it can be deduced that the probe does not use a public IP address to perform tests or communicate with the API, as it uses the same private IP address as the host PC.



Test type: ping

Target: 8.8.8.8

Private hostnames are not allowed.

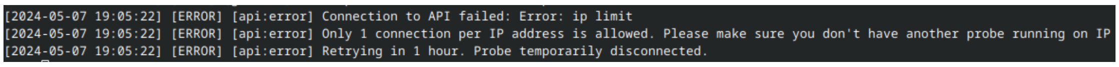
```
Content-Length: 159
Origin: https://www.jsdelivr.com
Referer: https://www.jsdelivr.com/
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: cross-site
Te: trailers

{
  "type":"ping",
  "inProgressUpdates":true,
  "target":"198.168.64.34",
  "limit":"10",
  "locations":[
    {
      "magic":"World"
    }
  ],
  "measurementOptions":{
    "packets":3
  }
}
```

Inspector

Request attributes	2
Request query parameters	0
Request cookies	0
Request headers	16

■ **Figure 5.1** Trying to scan a private IP address

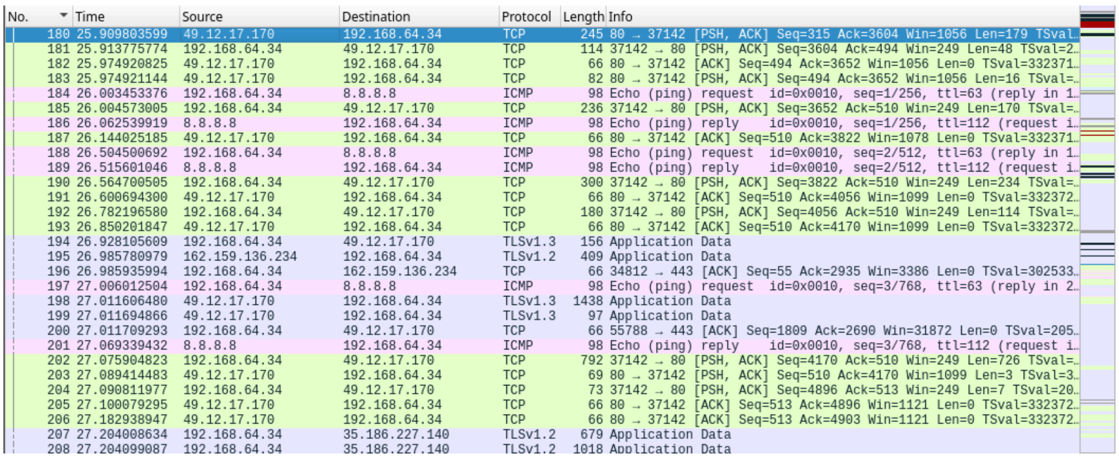


```
[2024-05-07 19:05:22] [ERROR] [api:error] Connection to API failed: Error: ip limit
[2024-05-07 19:05:22] [ERROR] [api:error] Only 1 connection per IP address is allowed. Please make sure you don't have another probe running on IP
[2024-05-07 19:05:22] [ERROR] [api:error] Retrying in 1 hour. Probe temporarily disconnected.
```

■ **Figure 5.2** Trying to run 2 Globalping probes on a single public IP address

HTTPS communication

Below is a packet capture of the probe performing a test after receiving a request from the API. As can be seen, not all of the transmitted data is secured.



```
No.  Time         Source           Destination      Protocol Length Info
 180 25.909803599 49.12.17.170     192.168.64.34    TCP       245 80 → 37142 [PSH, ACK] Seq=315 Ack=3604 Win=1056 Len=179 TSval…
 181 25.913775774 192.168.64.34    49.12.17.170     TCP       114 37142 → 80 [PSH, ACK] Seq=3604 Ack=494 Win=249 Len=48 TSval=2…
 182 25.974920825 49.12.17.170     192.168.64.34    TCP        66 80 → 37142 [ACK] Seq=494 Ack=3652 Win=1056 Len=0 TSval=332371…
 183 25.974921144 49.12.17.170     192.168.64.34    TCP        82 80 → 37142 [PSH, ACK] Seq=494 Ack=3652 Win=1056 Len=16 TSval=…
 184 26.003453376 192.168.64.34    8.8.8.8          ICMP       98 Echo (ping) request  id=0x0010, seq=1/256, ttl=63 (reply in 1…
 185 26.004573005 192.168.64.34    49.12.17.170     TCP       236 37142 → 80 [PSH, ACK] Seq=3652 Ack=510 Win=249 Len=170 TSval=…
 186 26.062539919 8.8.8.8          192.168.64.34    ICMP       98 Echo (ping) reply    id=0x0010, seq=1/256, ttl=112 (request i…
 187 26.144025185 49.12.17.170     192.168.64.34    TCP        66 80 → 37142 [ACK] Seq=510 Ack=3822 Win=1078 Len=0 TSval=332371…
 188 26.504500692 192.168.64.34    8.8.8.8          ICMP       98 Echo (ping) request  id=0x0010, seq=2/512, ttl=63 (reply in 1…
 189 26.515601046 8.8.8.8          192.168.64.34    ICMP       98 Echo (ping) reply    id=0x0010, seq=2/512, ttl=112 (request i…
 190 26.564700505 192.168.64.34    49.12.17.170     TCP       300 37142 → 80 [PSH, ACK] Seq=3822 Ack=510 Win=249 Len=234 TSval=…
 191 26.600694300 49.12.17.170     192.168.64.34    TCP        66 80 → 37142 [ACK] Seq=510 Ack=4056 Win=1099 Len=0 TSval=332372…
 192 26.782196580 192.168.64.34    49.12.17.170     TCP       180 37142 → 80 [PSH, ACK] Seq=4056 Ack=510 Win=249 Len=114 TSval=…
 193 26.850201847 49.12.17.170     192.168.64.34    TCP        66 80 → 37142 [ACK] Seq=510 Ack=4170 Win=1099 Len=0 TSval=332372…
 194 26.928105609 192.168.64.34    49.12.17.170     TLSv1.3   156 Application Data
 195 26.985780979 162.159.136.234  192.168.64.34    TLSv1.2   409 Application Data
 196 26.985935994 192.168.64.34    162.159.136.234  TCP        66 34812 → 443 [ACK] Seq=55 Ack=2935 Win=3386 Len=0 TSval=302533…
 197 27.006012504 192.168.64.34    8.8.8.8          ICMP       98 Echo (ping) request  id=0x0010, seq=3/768, ttl=63 (reply in 2…
 198 27.011606480 49.12.17.170     192.168.64.34    TLSv1.3  1438 Application Data
 199 27.011694866 49.12.17.170     192.168.64.34    TLSv1.3    97 Application Data
 200 27.011709293 192.168.64.34    49.12.17.170     TCP        66 55788 → 443 [ACK] Seq=1809 Ack=2690 Win=31872 Len=0 TSval=205…
 201 27.069339432 8.8.8.8          192.168.64.34    ICMP       98 Echo (ping) reply    id=0x0010, seq=3/768, ttl=112 (request i…
 202 27.075904823 192.168.64.34    49.12.17.170     TCP       792 37142 → 80 [PSH, ACK] Seq=4170 Ack=510 Win=249 Len=726 TSval=…
 203 27.089414483 49.12.17.170     192.168.64.34    TCP        69 80 → 37142 [PSH, ACK] Seq=510 Ack=4170 Win=1099 Len=3 TSval=3…
 204 27.090811977 192.168.64.34    49.12.17.170     TCP        73 37142 → 80 [PSH, ACK] Seq=4896 Ack=513 Win=249 Len=7 TSval=20…
 205 27.100079295 49.12.17.170     192.168.64.34    TCP        66 80 → 37142 [ACK] Seq=513 Ack=4896 Win=1121 Len=0 TSval=332372…
 206 27.182938947 49.12.17.170     192.168.64.34    TCP        66 80 → 37142 [ACK] Seq=513 Ack=4903 Win=1121 Len=0 TSval=332372…
 207 27.204008634 192.168.64.34    35.186.227.140   TLSv1.2   679 Application Data
 208 27.204099087 192.168.64.34    35.186.227.140   TLSv1.2  1018 Application Data
```

■ **Figure 5.3** Sample network traffic during a test

Rate limiting

The Globalping platform states a limit of 100 tests per hour for unauthenticated users and a limit of 200 tests for authenticated users per hour. In practice, none of these limits apply as even an unauthenticated user can perform thousands of measurements and what is more, at the moment there is no way to log in as an authenticated user.

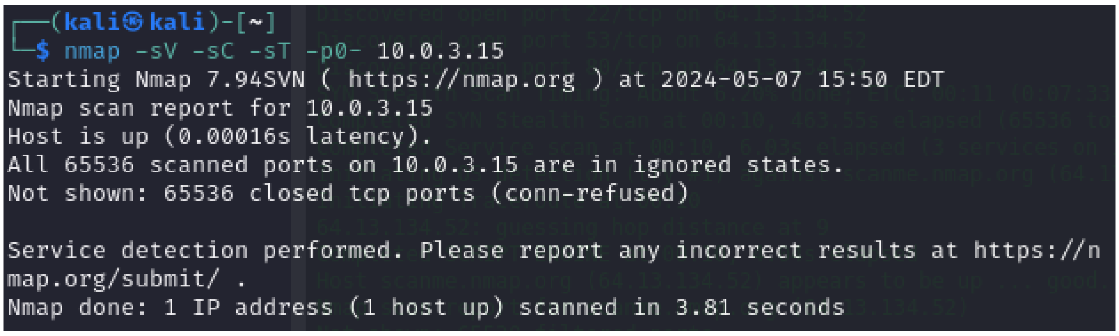
After clarifying this issue with the Globalping team, the authentication feature was confirmed as currently in development, and as such these limits are currently not enforced.

This may lead to an attacker abusing the Globalping platform to perform DOS attacks on other services.

Outgoing connections only

The probe does not establish incoming connections, as the probe initializes the connection to the API and there are no services to connect to on the probe.

- **No additional ports**
  Running a probe does not in fact open new ports on the host machine as can be proven by the Nmap scan 5.4 showcased below.



```
┌──(kali㉿kali)-[~]
└─$ nmap -sV -sC -sT -p0- 10.0.3.15
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-05-07 15:50 EDT
Nmap scan report for 10.0.3.15
Host is up (0.00016s latency).
All 65536 scanned ports on 10.0.3.15 are in ignored states.
Not shown: 65536 closed tcp ports (conn-refused)

Service detection performed. Please report any incorrect results at https://n
map.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 3.81 seconds
```

Figure 5.4 Port scan using Nmap

5.3 External dependencies

The external dependencies used by the application are secure, and there are currently no known vulnerabilities found in the versions, that are in use. However, the application uses dated versions of these libraries and thus, it is recommended to update these dependencies to their current versions. Below in Table 5.1 is a list of all outdated dependencies and their newest versions at the time of writing.

Name	Used version	Current version
Execa	7.2.0	9.0.2
Got	12.6.1	14.2.1
http2-wrapper	2.2.0	2.2.1
joi	17.11.0	17.13.1
socket.io-client	4.7.2	4.7.5
tldts	6.0.15	6.1.19
winston	3.10.0	3.13

Table 5.1 Table of assets

5.4 Source code analysis

The following section will cover the source code analysis of the Globalping probe.

5.4.1 index.ts

The class index takes care of initializing the probe after it is launched. It also handles establishing a connection with the API and managing errors. Additionally, the class manages the calls to the functions for running tests after receiving a request.

Let us break down these processes:

- **Initialization:** During initialization, the probe checks if it is on the latest version, if not, then the probe terminates itself and initiates a self-update mechanism. This process is accomplished by sending a GET request to the Globalping API and parsing the reply to obtain the new version of the probe. This is then compared to the current version and if a mismatch is detected, the self-update mechanism starts. During this, the versions are once again checked, the latest version of the probe is downloaded using `curl` and the files in the /app directory are replaced by these new ones. This process is a standard way of handling updates on applications running in containers and can be considered secure as the source is the Globalping API from where the new files are safely downloaded.

After this step is completed, the probe sets up periodic restarts and checks if the `unbuffer` binary is installed on the system and again if not, then it is downloaded and installed. After these steps are completed, the probe attempts to connect to the API.

- **Connection:** The module then handles the connection to the API. During this process, the node generates a random ID using the `randomUUID()` function from the `node:crypto` library. This value is then used as a unique identifier of the probe. At no point during the connection process does the API perform verification of the probe. This leaves the possibility of an attacker posing as a probe while using a malicious program to spoof the API. This connection is established using the WebSocket with the sw protocol, which creates an unencrypted connection.
- **Connection error handling:** In the case of a connection error, the probe tries to reconnect after an established timeout period. This helps in not flooding the API with connection requests if there is a problem with the probe.
- **Command handling:** After receiving a request to perform a test, the module calls the respective handler of the test and forwards the parameters to that handler.
- **Termination:** After receiving the `SIGTERM` message from the API, the module handles the graceful shutdown of the probe.

5.4.2 Commands

All of the commands except the HTTP command (which will be covered separately) are implemented in a similar fashion. First, the handler function takes the user-supplied data. The user input is then validated against a joi schema and fed to the function `argbuilder` to be turned into arguments for the binary, that will be executed. Finally, the binary is executed using the `execa` library, before parsing the output of the measurement and sending it back to the API.

This section deserves a closer look, as it takes, processes, and supplies user input to the underlying binary file. We can define the argument `options` of the run method as a source and the call to the `execa` function as a sink. Let us take a closer look at the flow of data between these two points.

We begin by analyzing the joi data validation schema 5.1, as it is the first place the data enters. We can see that most elements are properly validated with required types, default values, or even lists of allowed values. But the element `target` is not being validated in all commands, except html. Furthermore, when it comes to the `mtr` and `traceroute` commands we find that the `protocol` element is not validated. Finally, in the DNS command, we can find an unvalidated element `resolver`.

The user input is then forwarded to the `argBuilder` 5.2 function, where it is turned into arguments for the respective function. Additionally, this function adds a number of predefined arguments. All of this then gets returned as an array of all arguments.

If we return to the elements, that we defined as unvalidated in the paragraph before this one, we can observe that the unvalidated element `protocol`, from the `traceroute` command, is thrown out, if it does not equal TCP, thanks to the line:
``const port = options.protocol === 'TCP' ? [ '-p', `${options.port}` ] : [];``. The element `resolver` from the DNS command, and the element `protocol` from the `mtr` command are

```
const dnsOptionsSchema = Joi.object<DnsOptions>({
        type: Joi.string().valid('dns'),
        inProgressUpdates: Joi.boolean(),
        target: Joi.string(),
        resolver: Joi.string().optional(),
        protocol:
        ↪  Joi.string().valid(...allowedProtocols).optional().default('udp'),
        port: Joi.number().optional().default('53'),
        trace: Joi.boolean().optional().default(false),
        query: Joi.object({
                type:
                ↪  Joi.string().valid(...allowedTypes).optional().default('A'),
        }),
});
```

Listing 5.1 Example of a joi validation schema used in the DNS command.

forwarded as `@${options.resolver}`. and `--${options.protocol}` respectively. That leaves us with the element target, which is added to the array as is, without any changes.

```
export const argBuilder = (options: DnsOptions): string[] => {
        const protocolArg = options.protocol.toLowerCase() === 'tcp' ? '+tcp' :
        ↪  [];
        const resolverArg = options.resolver ? `@${options.resolver}` : [];
        const traceArg = options.trace ? '+trace' : [];
        const queryArg = options.query.type === 'PTR' ? '-x' : [ '-t',
        ↪  options.query.type ];

        .
        .
        .

        return args;
};
```

Listing 5.2 Example of argument parsing for the `dig` utility.

The command is then ran using the `execa` function with the `unbuffer` binary so that the output is received in one go. Below is an example with the `dig` binary.5.3.

It is important to note, that the user-supplied data is also validated on the API level, where the validation is much more thorough but we will not go into more detail as the API is out of scope for this analysis. But, this fact does not excuse the weaker authentication on the probe, even though it does not introduce a vulnerability, it violates the principle of defense in depth.

Continuing to the output handling. The application parses the output and sends the parsed data back to the API. During the parsing process, the application performs checks for private IP addresses using a `blocklist` implemented in the class `private-ip`. If a private IP address is detected, an error is returned to the API.

```
export const dnsCmd = (options: DnsOptions): ExecaChildProcess => {
        const args = argBuilder(options);
        return execa('unbuffer', [ 'dig', ...args ]);
};
```

■ **Listing 5.3** Example of executing a Linux binary.

5.4.2.1 http-command

The http command is singled out because, in contrast to other commands, that execute Linux binaries, this command is implemented directly in the source code. The structure is the same as the other commands.

To begin with, the user input is validated using a Joi schema with the correct level of strictness. However, unlike the other commands, there is no `argbuilder` function. Instead, the `urlBuilder` function takes the user-supplied data and builds the target URL using it.

Next, the data moves to the `httpCmd` function, which builds the HTTP request and sends it to the target. One possible improvement to this function would be to separate the building of the request into a separate function to keep the pattern more in line with the rest of the codebase.

It is interesting to note that the check if the target is a private IP address happens in the `DNS resolver`.

Finally, the implementation then continues with parsing the output. Again, unlike the other command implementations, the parsing is implemented in the `HttpCommand` class, instead of being implemented in a helper class as is the case with other commands.

5.4.3 Handlers

Handlers represent a group of functions to parse output from the `dig`, http, `mtr`, and `ping` commands. Each of these utilities has its dedicated parser, that properly handles any non-standard output states. The output from these binaries is parsed, converted to JSON, and finally sent back to the API. The content is parsed based on regular expressions. The only outlier is again the http utility, which instead of a parser implements a DNS resolver in the handler folder.

5.4.3.1 Sending data to the API

The classes progress-buffer-overwrite and progress-buffer are responsible for sending measurement data back to the API and the user. As their names suggest, these classes accumulate data before periodically sending it in predefined intervals after a timeout is triggered. This is helpful in not flooding the network with unnecessary messages. The timeout period is set sensibly and there are no security issues with the implementation.

5.4.3.2 Error handling and logging

Logs are created using the Winston library and each scope has its own logger. This helps the readability of the logs. The logs are only local and are not being sent to the API. Their structure is well-defined and readable. All of this is handled by the class logger.

When it comes to error handling the probe defines the class internal-error that extends the default error class. API errors are handled by the api-error-handler. And the execa errors are handled by the execa-error check. If an error occurs, it is well handled and logged.

5.5 Summary

In summary, the Globalping probe has the following vulnerabilities: Lack of authentication on the API, Insufficient input validation, Improper rate limiting, Insecure communication, and outdated dependencies. None of these pose a significant threat to the hosts of the probe, as the platform also validates input at the API, and the other vulnerabilities pose a bigger threat to the platform, than to the individual hosts. All of these vulnerabilities will be discussed in more detail in the next chapter. Other than that, the application shows a well thought out design with good security practices.

Chapter 6

Findings and mitigation

The findings of the security analysis of the Globalping platform are summarized in the following chapter. The security analysis consisted of validating the individual security claims made by the Globalping team, followed by a security analysis of the source code of the application. This analysis uncovered the following findings:

- Lack of authentication on the API
- Insufficient input validation
- Improper rate limiting
- Insecure communication
- Outdated dependencies

Under correct circumstances, these flaws could lead to a breach of confidentiality and integrity in the future.

For a more detailed description of these vulnerabilities, please refer to the rest of the chapter.

6.1 Lack of authentication on the API

The probe does not perform any kind of authentication upon establishing a connection with the API. This can lead to an attacker connecting a malicious program to the Globalping network, that pretends to be a probe while performing other actions.

While it is understood, that as an open network of probes, that gives anyone the freedom to contribute, this issue is difficult to mitigate. A possible alternative would be to implement manual authentication of probes, as is the case with RIPE Atlas, but that would lead to a more closed network, which conflicts with the values set by the Globalping team.

6.1.1 Scoring

- **Attack Vector:** Network
- **Attack Complexity:** High
- **Privileges Required:** None
- **User Interaction:** None

- **Scope:** Unchanged
- **Confidentiality** None
- **Integrity** Low
- **Availability** None

Supplying the values above into a CVSS calculator gives a score of 3.7, which is considered a vulnerability of low severity.

6.1.2 Remediation

As discussed above, mitigating this vulnerability is not possible without imposing additional restrictions on the network of probes, which is in direct conflict with the philosophy of the team at Globalping. As such the only recommendation is to remain vigilant and proactive in monitoring the behavior of the devices connected to the Globalping network.

6.2 Insufficient input validation

In the case of DNS, `mtr`, `ping`, and `traceroute` commands, the element target is not validated. There is no rule to ensure that the value supplied is an IP address or valid domain. Which means that, a potential attacker may abuse this vulnerability to inject and execute code on the probe.

Thanks to the data validation on the API this vulnerability does not have an impact on the security of the probe. Still, it is important to mention because it could possibly turn into an exploitable vulnerability in the future.

6.2.1 Scoring

- **Attack Vector:** Network
- **Attack Complexity:** High
- **Privileges Required:** None
- **User Interaction:** None
- **Scope:** Unchanged
- **Confidentiality** None
- **Integrity** None
- **Availability** None

As it was not possible to exploit this vulnerability during testing, this is only considered as an informative finding with a CVSS score of 0.

6.2.2 Remediation

It is recommended to implement expanded validation schemas for the DNS, `mtr`, `ping`, and `traceroute` commands. Mainly to validate the target element of the user-supplied data.

6.3 Improper rate limiting

The platform fails to implement rate limiting in the specified range. The current limit is set at100 000 tests instead of the claimed 100 tests for unauthenticated users or 200 tests for authenticated users. While it is understood, that this finding is out of the scope of this analysis, as it involves the Globalping API and not the probe, I felt that it was important to mention this finding.

Another caveat is the fact, that authentication for the users is still a feature in development but the claimed limit should represent the actual limit that is set.

6.3.1 Scoring

- **Attack Vector:** Network
- **Attack Complexity:** Low
- **Privileges Required:** None
- **User Interaction:** None
- **Scope:** Unchanged
- **Confidentiality** None
- **Integrity** None
- **Availability** Low

Calculating the score of this vulnerability gives us a value of 5.3, which makes this vulnerability of medium severity.

6.3.2 Remediation

It is recommended to change the limit to the claimed value or edit the documentation to reflect the real state.

6.4 Insecure communication

The application uses an insecure WebSocket protocol to establish a connection to the API.

6.4.1 Scoring

- **Attack Vector:** Network
- **Attack Complexity:** Low
- **Privileges Required:** None
- **User Interaction:** None
- **Scope:** Unchanged
- **Confidentiality** Low
- **Integrity** None
- **Availability** None

As the vulnerability has an impact on the confidentiality of the information, it is per the breakdown above, assigned a score of 5.3, which is considered of medium severity.

6.4.2 Remediation

Use wss instead of ws when establishing a connection to the API, as wss establishes a TLS encrypted connection.

6.5 Outdated dependencies

The application uses outdated dependencies, which may lead to a potential introduction of a vulnerability into the probe, through insufficient patching.

6.5.1 Scoring

- **Attack Vector:** Network
- **Attack Complexity:** High
- **Privileges Required:** None
- **User Interaction:** None
- **Scope:** Unchanged
- **Confidentiality** None
- **Integrity** None
- **Availability** None

As none of these dependencies are vulnerable in their current state, this finding is considered informative in nature and is assigned a score of 0.

6.5.2 Remediation

Update all of the dependencies to their latest version and periodically check for updates to these dependencies.

Chapter 7
Discussion

The analysis of the Globalping probe uncovered the following security vulnerabilities: Lack of authentication on the API, Insufficient input validation, Improper rate limiting, Insecure communication, and outdated dependencies. While these may sound severe, for the probe, in the context of the platform as a whole they do not pose a notable threat. The insufficient input validation is mitigated on the API level and this finding is more informative in nature as in suggesting best practices of defense in depth. The authentication on the API is a design choice as it is complicated to implement in an open-source application. When it comes to the Insecure communication and Outdated dependencies, these issues were already fixed in a newer version of the probe during the testing process. As this update came out during the testing process, these findings remained in the listing of vulnerabilities, although they are already mitigated. Finally, the improper rate limiting is a misleading entry in the documentation, as the documentation includes features, not yet implemented in the probe, and still remains an issue.

Taking all of this information into account and combining it with the outcome of the whole thesis, it can be said that hosting the tested version of the probe should be secure. There are minor issues for the development team to stay on top of, but these do not pose a threat to the end users.

When it comes to future work on this topic, it should be noted that security reviews should be a periodic action happening at least once a year. The Globalping platform would also benefit from penetration testing of not only the probe but also the API.

Chapter 8
Conclusion

The goal of my thesis was to perform a security analysis of the Globalping probe. To accomplish this goal, first, it was crucial to get familiar with the Globalping platform as described in the first chapter. Then to gain further perspective I have also taken a look at some of the competitors to the Globalping platform and the security measures, that they implement on their platforms. Armed with this knowledge I was able to complete a threat model of the Globalping platform, thus summarizing the biggest threats to the platform and its users, helping me focus on the most important parts of the Globalping probe during the analysis. Finally, I performed a security code review of the source code of the Globalping probe to identify any potential vulnerabilities and recommended possible mitigation.

During the analysis of the Globalping probe, I did not manage to uncover any severe vulnerabilities in the application. The few vulnerabilities, that were identified, are mostly informative in nature. As a result of these steps, I produced a security report on the Globalping probe, which is a valuable resource for both the developers of the Globalping platform and also for the users who aim to host a probe themselves.

However, it is important to note that this conclusion is not final, as security is an ongoing process. Therefore, it is crucial to ensure that this analysis is not a one-time event but a regular part of the development cycle of the Globalping probe.

Regarding my personal objectives, I believe that I have met the goals that I set out to accomplish. I am confident that the hands-on experience will prove useful in the future.

Bibliography

1. *Globalping - Internet and web infrastructure monitoring and benchmarking — jsdelivr.com* [online]. jsDelivr. Available also from: `https://www.jsdelivr.com/globalping`. [Accessed 09-03-2024].
2. *GitHub - jsdelivr/globalping: A global network of probes to run network tests like ping, traceroute and DNS resolve — github.com* [online]. jsDelivr. Available also from: `https://github.com/jsdelivr/globalping`. [Accessed 05-05-2024].
3. *GitHub - jsdelivr/globalping-probe: The globalping probe code that runs on your hardware and connects to the global community network of probes — github.com* [online]. jsDelivr. Available also from: `https://github.com/jsdelivr/globalping-probe`. [Accessed 05-05-2024].
4. *RIPE Atlas and Globalping: Choosing the Right Network Measurement Platform | HackerNoon — hackernoon.com* [online]. Available also from: `https://hackernoon.com/ripe-atlas-and-globalping-choosing-the-right-network-measurement-platform`. [Accessed 23-04-2024].
5. *RIPE Atlas - RIPE Network Coordination Centre — atlas.ripe.net* [online]. RIPE NCC. Available also from: `https://atlas.ripe.net/`. [Accessed 09-03-2024].
6. *RIPE Atlas and Globalping: Choosing the Right Network Measurement Platform — dev.to* [online]. Available also from: `https://dev.to/globalping/ripe-atlas-and-globalping-choosing-the-right-network-measurement-platform-487f`. [Accessed 09-03-2024].
7. *RIPE Atlas docs | Security and Privacy | Docs — atlas.ripe.net* [online]. RIPE NCC. Available also from: `https://atlas.ripe.net/docs/faq/security-and-privacy.html`. [Accessed 23-04-2024].
8. *Threat Modeling | OWASP Foundation — owasp.org* [online]. OWASP. Available also from: `https://owasp.org/www-community/Threat_Modeling`. [Accessed 19-03-2024].
9. DOMARS. *Threat Modeling for Drivers - Windows drivers — learn.microsoft.com* [online]. Microsoft. Available also from: `https://learn.microsoft.com/en-us/windows-hardware/drivers/driversecurity/threat-modeling-for-drivers#the-dread-approach-to-threat-assessment`. [Accessed 13-04-2024].
10. *Static Code Analysis | OWASP Foundation — owasp.org* [online]. OWASP. Available also from: `https://owasp.org/www-community/controls/Static_Code_Analysis`. [Accessed 14-04-2024].
11. *NVD - Vulnerability Metrics — nvd.nist.gov* [online]. NIST. Available also from: `https://nvd.nist.gov/vuln-metrics/cvss`. [Accessed 27-04-2024].

Content of annexes

```
readme.txt ............................. a brief description of the content of the medium
src
  thesis ..................................................... source thesis in LaTeX format
text ........................................................................ the text of the thesis
  thesis.pdf ........................................... text of the thesis in PDF format
```