

A Thousand Eyes

Global Network Monitoring Solution

Author: [REDACTED]
Team: [REDACTED]
Reviewer: [REDACTED]
Version: [REDACTED]
Created On: [REDACTED]
Last Updated: [REDACTED]
Ticket Link: N/A

This document contains a case study that I once completed as part of an interview process. Some information is censored so that I can share it here.

Contents

1	Introduction	2
1.1	Overview	2
1.2	Goals and Requirements	2
1.3	Out-of-Scope and Future Goals	2
1.4	Assumptions	2
2	Solution	2
2.1	Methodology	2
2.2	Deployment	4
2.3	Architecture	4
3	Pros, Cons, and Alternatives	6
3.1	Limitations and Tradeoffs	6
3.2	Scanning Alternatives	6
3.3	Data Alternatives	6
3.4	Migration Plan	6
4	Further Discussion	7
4.1	Failure Events	7
4.2	Legal Concerns	7
4.3	Open Questions	7
A	Revision History	8
B	List of Terms	9
C	List of Tables	10
C.1	Table 1: Selected Masscan Arguments	10
C.2	Table 2: Significant Values	10
D	List of Figures	11
D.1	Figure 1: Architecture Overview	11
D.2	Figure 2: Sample Masscan Config	11
D.3	Figure 3: Lens Pseudocode	12
D.4	Figure 4: Data Schema	12
E	Related Work	13
F	References	14

1 Introduction

1.1 Overview

The [REDACTED] team is tasked with implementing a solution to periodically record all available ports and services for all IPv4 and IPv6 addresses. There are no third-party products which directly provide this data and [REDACTED] likewise does not have a solution in place, so this will be an entirely bespoke project. Solving this problem will allow us to improve our detection of hackers and bots moving forward, rendering [REDACTED] more compelling to potential customers and hopefully boosting metrics such as **ARPU** and **ARR**.

I propose that we center our solution around the open-source software, **Masscan**. Masscan is a port scanner similar to **Nmap** which operates on the scale of the entire internet. We will develop an internal microservice to collect the data and update [REDACTED] with the appropriate information, then deploy this microservice at scale using virtual machines. This is not a perfect solution and relevant considerations will be discussed later in this document.

The solution is to be named **A Thousand Eyes**.

1.2 Goals and Requirements

The purpose of this project is to record all available ports and services for all IP addresses at regular intervals. Once implemented, this should be a hands-off, automated process. We can address the task in three steps:

1. Determine which IP addresses to target (specifically regarding IPv6).
2. Get data on addresses from Step 1 with Masscan.
3. Add resultant data to [REDACTED].

The greatest difficulty here will be collecting the data in a reasonable timeframe. Otherwise, we will need to ensure that the data is provided in a usable format when added to [REDACTED].

1.3 Out-of-Scope and Future Goals

To reiterate, we are just collecting data here. There is no intent to make any assumptions from the data, nor is there any intent to manipulate this data in any way beyond providing it in a useful format. Once we have addressed the issue of data collection, we can consider further utilizing this data as a feature set for machine learning models, attack investigations, and classification of IP addresses into clusters to improve user experiences.

1.4 Assumptions

- [REDACTED] is SQL-like and can be programmatically accessed to update data.
- [REDACTED] supports JSON data or can somehow accomodate JSON-like data.
- Data in [REDACTED] is readily available to [REDACTED] engineers.
- We are able to divert a sufficient amount of [REDACTED]'s infrastructure to this project.

2 Solution

2.1 Methodology

To keep terminology simple, we will refer to the entire solution as A Thousand Eyes, with each instance of a virtual machine running our software known as an **Eye**. The software running on each eye will be called the **Lens** and will be identical across all Eyes. We will design the Lens to automatically execute scans of the entire internet, then distribute it across a specified number of eyes at various COLOs. Masscan has built-in support for **Sharding**, so distributed execution alone is relatively straightforward. The real difficulty here is in addressing the *entire* internet. To scan the entire IPv4 address range (2^{32} addresses) takes just minutes for a single machine, but to scan the entire IPv6 address

range (2^{128} addresses) is unfeasible, even with ██████'s impressive infrastructure available. Resolving this issue is ultimately left as an open question, but we realistically do not need to check all possible IPv6 addresses. With careful targeting, gathering the required data should be doable. See Section 4.3: Open Questions for more thought on this issue. The exact number of Eyes needed will depend on the target runtime and the amount of IP addresses scanned, but for now we can assume there will be 1,000 instances.

Masscan supports sufficient arguments to meet the requirements of A Thousand Eyes, all of which can be stored in a lightweight configuration file:

Arguments	
Name	Description
rate	Decimal number representing desired scan rate in packets per second
output-format	Format of scanned data output (e.g. JSON)
output-filename	name of file to which scanned data will be output
ports	Ports to scan for each host, can be range (0-65535)
range	IP address range to scan, can include multiple ranges
excludefile	File listing addresses <i>not</i> to scan
shard	Shard designation for this scan, given as a fraction representing the shard index (e.g. 1/2)

Table 1: Selected Masscan Arguments

Storage of the configuration file is addressed in Section 2.2: Deployment. We will still need to specify a shard number at the per-Eye level. In order to facilitate this in a scalable manner, we can use ██████ to store unique identification numbers for each shard, perhaps based on hostname or some other unique string. These identification numbers can then be accessed and used to modify the configuration in memory during execution.

The Lens will initially read all required configuration data into memory, formatting as necessary. It will then immediately execute a shell command to run its designated portion of the internet-scale scan with Masscan. No matter how many Eyes we deploy, this will not be instantaneous. Each Eye can reasonably be expected to handle up to 300 kpps of traffic. According to the developer, this is sufficient for one host to scan the entire IPv4 address range in approximately $3\frac{1}{2}$ hours. Multiplying by 1,000, I estimate that we can scan approximately 130 billion addresses per hour. Our targeted IPv6 address range should fall well below this number, so we will proceed under the assumption that runtime is up to an hour and that this is reasonable given the scope of the project. It is possible that we may narrow our target address range and further reduce runtime to as low as fifteen minutes later on.

Significant Values	
Name	Value
Scan Rate	300 kpps
Ports per Host	65535
Total COLOs	270
Network Capacity	142 Tbps
Total Eyes	Variable
IPv4 Addresses	4.3×10^9
IPv6 Addresses	3.4×10^{38}

Table 2: Significant Values

When the scan does finish, output can be provided in several formats. I recommend JSON for readability and ease of use, although we may wish to consider other options as discussed in Section 3.1: Limitations and Tradeoffs. In any case, the Lens will proceed to parse the output, remove unnecessary data, aggregate results by host, and finally upload the data to ██████. Progress is to be logged and reported at each step of this process.

2.2 Deployment

When considering deployment, we assume the software has already been built and fulfills the needs of our team as described. In order to maintain scalability and adaptability, I advise that we establish a small package to be stored in ██████'s internal repositories (or a new internal repository) and mirrored at all COLOs. This package will include the static Masscan configuration file as well as scripts to handle deployment and rollback of individual Eyes. When we narrow down our target IPv6 address range, we will add an "excludefile" configuration file to this repository for use with Masscan.

The deployment script will first install Masscan, then copy our Lens software and configuration files into a sensibly chosen directory on the Eye and add that directory to \$PATH. At this point, the Lens will be scheduled to scan at an arbitrary interval with **Cron**. We will likely want to pull this interval from ██████ in order to maintain consistency and scalability. Once all of the software is prepared, the deployment script will run a series of unit tests. Should these tests fail, descriptive alerts will be thrown using our existing internal monitoring tools. When tests pass, the Eye will be integrated into our existing infrastructure monitoring systems under an appropriate label.

The rollback script will remove the \$PATH entry, delete the directory containing the Lens, and then uninstall Masscan. It will then run some brief tests to ensure the Eye was properly "reset" before running a system update. Finally, the script will communicate to our monitoring system that this Eye has been decommissioned and returned to Cloudflare's pool of available hardware. This script is to be run on uninstall for any given Eye.

As each Eye is nothing more than a virtual machine running lightweight internal software, it is sufficient to work with our existing monitoring and alerting systems rather than build a dedicated solution for this aspect of A Thousand Eyes. Depending on how well we can integrate this project with our other systems, we may still want to consider a dedicated solution. Namely, we should have the ability, to deploy, deactivate, and reinstall eyes in an accessible manner. We should also be able to run and automate tests, and to check configurations on demand.

2.3 Architecture

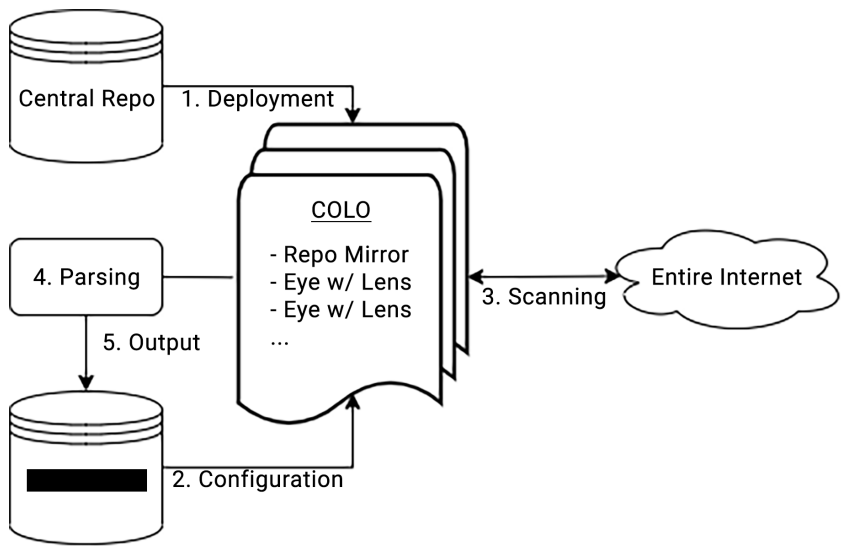


Figure 1: Architecture Overview

Development is done in a central repository. From there, this repository is mirrored at all COLOs. Regarding deployment, we simply instantiate a new virtual machine, install the deployment package, and run the script to deploy. The Eye is then ready for use.

As the package is lightweight by design, it makes sense to mirror it even at potentially unused COLOs in case we want to increase the Eye count in the future. Configuration files are to be stored on [REDACTED] and accessed by the Lens at runtime. A sample configuration file is provided in Figure 2.

With all COLOs properly prepared, each Eye will scan at the same time as specified in Cron. Recall that we added a task to Cron when the deployment script was run on each Eye, so this is already taken care of. Furthermore, the Eyes operate independent of each other. By design, Masscan shards will scan separate portions of a total range according to their shard number. Since each Eye can thus scan the appropriate IP addresses and upload to [REDACTED] asynchronously, they actually do not need to communicate with each other at all.

When scanning is complete, the Lens software proceeds to parse the data and deliver the result in JSON format. Each JSON output is to contain a list of host entries, where each host's value is a list of ports mapped to services. This is not the most memory-efficient JSON format, but it is the most convenient for our use. Pseudocode and JSON schema are provided in Figures 3 and 4.

```
# Sample Config

# Target
rate = 300000.00
ports = 0-65535
range = 0.0.0.0/0, 2000::/3
excludefile = exclude.txt
shard = 1/1000

# Output
output-format = JSON
output-filename = output.json
```

Figure 2: Sample Masscan Config

Lens Pseudocode

```
- Monitoring: set status `configuring`
1. Read masscan.conf into memory
2. Read $shard into memory (shard ID)
3. Insert `shard = $shard` into masscan.conf

- Monitoring: set status `scanning`
4. Exec `masscan -c masscan.conf` (async)
   (Wait for Step 4 to finish)

- Monitoring: set status `parsing`
5. Read output.json into memory
6. Parse and reformat output
7. Save formatted output to a file

- Monitoring: set status `pushing`
8. Push output to [REDACTED] (async)
   (Wait for Step 8 to finish)

- Monitoring: set status `pending`
```

Figure 3: Lens Pseudocode

```
[
  {
    "host": [
      {
        "port": "service"
      },
      ...
    ]
  },
  ...
]
```

Figure 4: Data Schema

Finally, data is sent to [REDACTED]. As each Eye scans the same amount of IP addresses at the same rate, all results from the entire port scanning procedure should be delivered more or less at the same time, excepting network issues external to this process. At this point we have reached the end of this project's scope and we may manipulate the data as we please.

3 Pros, Cons, and Alternatives

3.1 Limitations and Tradeoffs

The proposed solution is elegant and involves minimal time in development. This will allow us to quickly bring the product into production, but it comes at a cost.

Logistics: Implementation will require a large number of physical servers to power our virtual machines (Eyes). These servers will need to be distributed around the globe in order to reduce the possibility of regional network strain. In a time of lengthening shipping delays, we may need to wait to move forward if we are low on hardware. There is also a concern regarding costs, although that is beyond the scope of this document.

Performance: [REDACTED]'s unique infrastructure may minimize latency, but we are still dealing with a massive amount of data. Setting aside runtime, the suggested JSON schema is not as compact as possible and will require substantial traffic between each Eye and our database.

Monitoring: I have deemed building a monitoring solution outside the scope of this project, but that means we will need to deal with our existing monitoring systems and their respective limitations. If we cannot monitor all of the information we want, we may need to consider building a dedicated monitoring system just for the servers involved.

3.2 Scanning Alternatives

I have found two open-source tools similar to Masscan: Zmap and Smap, which is powered by the **Shodan** search engine. We could also directly interface with Shodan, although this would complicate development and increase runtime (Shodan enforces a rate limit of one request per second on all plans). Lastly, there is the option to build this part of the solution entirely from scratch. Masscan proved to be the fastest option during testing, although the difference was insignificant. Where Masscan excels however is documentation and support.

3.3 Data Alternatives

As stated previously, the JSON schema could be made more compact. We could also output the raw JSON which will be even more efficient, or we could use a different output format. Masscan supports XML, list, and binary output in addition to JSON, with binary being the most efficient. The expanded JSON format was chosen to maximize code clarity and data accessibility for future use. We may need to revisit this depending upon performance.

3.4 Migration Plan

Regarding software, I see two "next best" options. We could revisit the Shodan API and see how much of an impact we face, or we could fork Masscan and tweak it to fit our use case. Taking into account time spent building the solution, it would be better to attempt working with Shodan first. In terms of data, we already have the ability to reduce the required storage capacity as discussed in the previous subsection. This will of course require further consideration if we significantly alter our software approach, for example by abandoning Masscan for Shodan.

4 Further Discussion

4.1 Failure Events

There are a few different ways in which the solution may fail:

- Database Failure: A failure of [REDACTED] would be of grave concern to [REDACTED] as a whole, although in our case the scanning software will still function. We would need to wait for the database to come back online, or change our output destination to a new alternative. Given the steps already taken to secure [REDACTED], this is a very unlikely scenario.
- Masscan Failure: If Masscan is broken or compromised, we will need to consider the Migration plan proposed in Section 3.4: Migration Plan. We can then proceed with development as described in that section.
- Any failure events which I have not yet thought of, which we should discuss as a team.

4.2 Legal Concerns

We will need to abide by the licenses under which any integrated open-source software is made available. In the case of Masscan and the alternative Smap, that is the AGPL-3.0 license. Zmap, our other Masscan alternative, is licensed under the Apache-2.0 license. Shodan is not open source and I could not locate Terms of Service or equivalent information during my research. These licenses should not be a factor when it comes to our internal use case.

The act of port scanning at scale is, to my knowledge, legal and is therefore not a concern here. We are likewise not considering any privacy concerns, as all information accessed is public.

4.3 Open Questions

There remain some questions which must be answered during the development of this solution.

1. Have efforts previously been made to address the issue of mass port scanning? If so, why have they failed? An answer will help us to avoid mistakes which have already hurt previous attempts at a solution.
2. What is the maximum acceptable runtime and how long do we wish to wait between scans? Answering this will help guide development, especially regarding the total number of IPv6 addresses targeted by our software.
3. We still need to determine a suitable subset of the IPv6 address range to target. This will depend in part on the answer to the previous question, but there are some obvious ways in which we could lower this number:
 - Only one eighth of the total IPv6 address space is currently in use, with the rest reserved for future use. This reduces the total address count to 4.3×10^{29} .
 - We can also exclude governments and large corporations which would likely block our requests anyway.

Appendices

A Revision History

Version	Date	Author	Description

B List of Terms

A Thousand Eyes The name given to the proposed solution for mass automated port scanning, with each “Eye” looking through the “Lens” at a designated portion of targeted IP addresses

ARPU Average Revenue Per Unit: profitability based on income generated at the per-user level

ARR Annual Recurring Revenue, a useful metric when considering the project’s value

COLO A data center in use by [REDACTED]

Cron A task schedule built into most Linux systems

Eye A single virtual machine running our software as part of the collective solution

Lens Software running on each “Eye” to facilitate mass port scanning

Masscan An open-source network-scanning utility similar to nmap, but operating on a much greater scale at rates of up to 1,500,000 packets per second

Nmap A network scanning utility which makes information on ports and services readily available for a given IP address

Sharding Distributing workload across multiple “shards” (for our purposes, virtual machines), a feature which is built into Masscan

Shodan Search engine and API providing information on ports and services in use by IP addresses

C List of Tables

C.1 Table 1: Selected Masscan Arguments

Arguments	
Name	Description
rate	Decimal number representing desired scan rate in packets per second
output-format	Format of scanned data output (e.g. JSON)
output-filename	name of file to which scanned data will be output
ports	Ports to scan for each host, can be range (0-65535)
range	IP address range to scan, can include multiple ranges
excludefile	File listing addresses <i>not</i> to scan
shard	Shard designation for this scan, given as a fraction representing the shard index (e.g. 1/2)

From Page 3

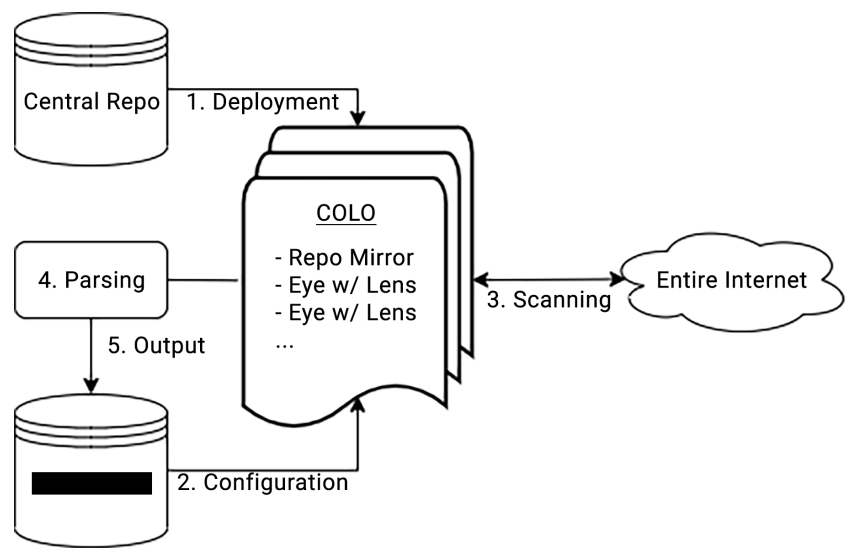
C.2 Table 2: Significant Values

Significant Values	
Name	Value
Scan Rate	300 kpps
Ports per Host	65535
Total COLOs	270
Network Capacity	142 Tbps
Total Eyes	Variable
IPv4 Addresses	4.3×10^9
IPv6 Addresses	3.4×10^{38}

From Page 3

D List of Figures

D.1 Figure 1: Architecture Overview



From Page 4

D.2 Figure 2: Sample Masscan Config


```
# Sample Config

# Target
rate = 300000.00
ports = 0-65535
range = 0.0.0.0/0, 2000::/3
excludefile = exclude.txt
shard = 1/1000

# Output
output-format = JSON
output-filename = output.json
```

From Page 5

D.3 Figure 3: Lens Pseudocode



Lens Pseudocode

- *Monitoring: set status `configuring`*
 1. Read masscan.conf into memory
 2. Read \$shard into memory (shard ID)
 3. Insert `shard = \$shard` into masscan.conf
- *Monitoring: set status `scanning`*
 4. Exec `masscan -c masscan.conf` (async)
(Wait for Step 4 to finish)
- *Monitoring: set status `parsing`*
 5. Read output.json into memory
 6. Parse and reformat output
 7. Save formatted output to a file
- *Monitoring: set status `pushing`*
 8. Push output to [REDACTED] (async)
(Wait for Step 8 to finish)
- *Monitoring: set status `pending`*

From Page 5

D.4 Figure 4: Data Schema



```
[
  {
    "host": [
      {
        "port": "service"
      },
      ...
    ]
  },
  ...
]
```

From Page 5

E Related Work

This section is left blank due to my limited permissions, but included to demonstrate that we should consider how the project relates to work done by different teams. It is important to be aware of such information to enable natural knowledge transfer between teams and to help tackle related issues as they arise.

F References

IPv6 Wikipedia Page: https://en.wikipedia.org/wiki/IPv6_address

Masscan Talk (Defcon): <https://www.youtube.com/watch?v=UOWexFaRylM>

Masscan on GitHub: <https://github.com/robertdavidgraham/masscan>

Smap on GitHub: <https://github.com/s0md3v/Smap>

Zmap on GitHub: <https://github.com/zmap/zmap>

Shodan Search Engine: <https://www.shodan.io/>