# dpacket
# Quick Start Guide

# Contents

## Introduction

Dpacket (formerly known as anonymizer) is a command-line utility for anonymizing, discovering, and doing light indexing of packet capture files. In this context, anonymization means protecting identifying information about network endpoints and their users.

For anonymization, Dpacket strives to do a good job at protecting sensitive data while preserving the analytical value of the original PCAP. There is no one-size-fits-all approach, no perfect tool or technique, and different packet artifacts are important to different folks. So, another main design goal is to provide flexibility and control over how anonymization tasks are performed.

Dpacket assumes sensible defaults and the only required argument is the source PCAP file. If you choose to omit all other flags, Dpacket will perform full anonymization of MAC Addresses and IP Addresses found in the Ethernet and IP headers respectively. The resulting output file (the *generated* trace) is generally suitable for common PCAP sharing use cases. However, Dpacket includes many other command-line flags dealing with PCAP data transformation, reporting and analysis.

## Basic Usage

Dpacket can be run in either **anonymize** or **discover** mode. Each mode is enabled with the corresponding command. Using dpacket for basic IP anonymization is simple - download the executable for your operating system and run the `dpacket anonymize` command, passing the `-source` flag to specify the source PCAP file you wish to anonymize. For example:

```
dpacket anonymize -source my_sensitive.pcap
```

By default, without any configuration flags set, ***Dpacket will anonymize the IP addresses and MAC addresses found in every Ethernet Frame and IP header read from the source PCAP***. This is done by encrypting each unique IP address and replacing the original value with the encrypted value throughout the trace. Each unique MAC address is replaced with a pseudo-randomly generated value. IP and Transport header checksums will also be recalculated as needed but no other changes are made to any frame or packet.

This default configuration covers many use cases, but Dpacket offers many other features to make the outcome more concise and effective. To see the full list of supported features and additional usage information use the `-h or -help` flag. We also cover that in more detail below.

If you're looking to get started quickly, follow these simple steps:

**1. Download the software binaries, documentation, and sample PCAPs:**

Dynamite Analytics has set up a temporary webpage for the DOE PCAP Dpacket project to allow the project reviewers to anonymously download the software and documentation. This page will be eventually replaced with the official software download that will require user registration.

PCAP Anonymizer (now Dpacket) project page: https://dynamite.ai/anonymizer-project

- Software binaries are available for Linux and macOS
- Sample PCAP files can be download from PacketTotal

**2. Set executable permissions on the binary:**

```
mv dpacket-linux-v0.1.5.bin dpacket ; chmod +x dpacket
```

**3. On macOS you may need to disable the verified app check:**

```
sudo spctl --master-disable
```

**4. Run the binary against a PCAP file:**

```
./dpacket anonymize -source my_sensitive.pcap
```

dpacket writes log messages to the console providing information on its configuration, tasks, and results. Optionally, logging to file can also be enabled, in which case dpacket will write detailed information about packet analysis and anonymization activities to a local log file. File logging is disabled by default but can be enabled with the `-log-file <log_level>` flag. Similarly console logging can be adjusted or disabled with the `-log-console <log_level>` flag.

When dpacket is executed as described above, a new, anonymized, capture file is created in the current directory named, my_sensitive-anonymized.pcap.

```
pktlab> dpacket anonymize -source my_sensitive.pcap
2023-12-06T14:48:17-05:00 [INFO] dpacket is starting up | command=anonymize
2023-12-06T14:48:17-05:00 [INFO] running anonymize on pcap file | command=anonymize filename=my_sensitive.pcap
2023-12-06T14:48:17-05:00 [INFO] opened pcap file for processing | command=anonymize filename=my_sensitive.pcap module=packet
2023-12-06T14:48:17-05:00 [INFO] created output pcap file | command=anonymize filename=/tmp/demo/dpacket/my_sensitive-anonymized.pcap module=packet
2023-12-06T14:48:18-05:00 [INFO] anonymize complete, took 0.164519s | command=anonymize
2023-12-06T14:48:18-05:00 [INFO] printing report | command=anonymize

capture_duration: 4m58.505344s
ip_layer_not_found: 18
mapped_ip_addresses: 222
source_md5: 16cf39fc81c1f8e3d0324f8dd860e569
output_file: /tmp/demo/dpacket/my_sensitive-anonymized.pcap
first_packet: 2011-01-25T13:52:22.484409-05:00
last_packet: 2011-01-25T13:57:20.989753-05:00
packets_analyzed: 14261
packets_ip_anonymized: 14243
packets_read: 14261
packets_written: 14261
source_file: /tmp/demo/dpacket/my_sensitive.pcap
total_runtime: 0.164519s

2023-12-06T14:48:18-05:00 [INFO] dpacket is shutting down | command=anonymize
```

## Best Practices

We have begun to refine sound practices for using Dpacket and sharing network trace files publicly in general. Preparation is key, and to prepare a PCAP file for sharing helps reduce risk of information leakage but it does not eliminate all risk. There is also an inverse relationship between data anonymization and the analytical value – or the legitimacy and usefulness - of the generated trace. As more data obfuscation protections are applied – in most cases - the analytical value of the generated trace is reduced.

For these reasons, we recommend you be as concise as possible when selecting traffic to be shared, and only applying protections you deem necessary.

- **Ensure you have authority and permission to capture and share traffic** – You must ensure you have permission to capture and share network traces before doing so. Network traffic traces contain a trove of sensitive information, in various forms, and are owned by the organization within which they are captured. Only sharing what you have the authority and permission to share, helps limit your liability and reduce the risk of unintentional information leakage.

- **Do not reuse secret keys** – While convenient, the reuse of secret keys increases the potential for an adversary to successfully decrypt encrypted values. Using randomly generated keys helps ensure a given key is only useful in decrypting a single trace file.

- **Use a source BPF filter** – Use capture filters or Dpacket's BPF option to restrict packets read from the source trace file. This is one of the most effective ways to reduce the risk of accidental information leakage. By only including the traffic that is intended to be shared, no other traffic poses a risk.

- **Target IP Address Anonymization** – Applying IP anonymization to all IP addresses can have adverse effects on the fidelity of the generated trace file. Under most circumstances, it is sufficient to protect IP addresses under one's ownership and control and leave others unmodified. This helps preserve more analytical value in the generated trace file while still protecting the identity of assets of interest.

- **Remove VLAN Headers** – Unless specifically needed, VLAN headers and ID's they contain provide little value or context to researchers working with an anonymized trace file. While they in themselves do not pose significant identity exposure risk, adversaries can use VLAN information to map a victim's network environment. Unless otherwise needed, we recommend you remove VLAN headers from anonymized packets to be shared using the `-vlan remove` flag or equivalent option.

- **Remove ICMP, ARP and other control messages** – Address resolution and control protocol messages pose elevated risk for privacy and asset identity leakage and should be discarded if not specifically required to satisfy a particular use case. By using restrictive BPF filters as described above Dpacket can prune unwanted messages from a generated trace file, greatly reducing the risk of identity and sensitive information leakage.

This is not an exhaustive list, but a few recommendations that are applicable to most use-cases. Of course, anonymization needs can vary widely depending on the intended use of the anonymized trace and Dpacket provides an extensive interface for customizing its anonymization features.

## Generate and Customize Options File

Dpacket provides many command-line flags as well as a customizable configuration file alternative, referred to as the *Options* file.

The easiest way to produce an Options file is to have Dpacket do it for you by passing the `-store-options` flag. As with before, with no other flags set, Dpacket will generate and write a YAML formatted options file like the one shown below - note the `-source` flag is *always* required:

```
data_link:
  mac_addresses: full
  vlan_ids: ignore
  arp_messages: ignore
network:
  ip_addresses: full
  ip_length: false
  icmp_messages: ignore
  preserve_all_special: false
  preserve_loopback: false
  preserve_multicast: false
  preserve_broadcast: false
  preserve_linklocal: false
  ip_tunnels: false
key: 7de5c6ffdf191271f524d4f4ba73413f
store_mappings: false
source_file: /tmp/demo/dpacket/my_sensitive.pcap
output_directory: /tmp/demo/dpacket
output_file: my_sensitive-anonymized.pcap
output_format: pcap
fix_checksums: true
pcapng_annotate: false
log_level: disabled
```

Open the YAML file in your favorite text editor to customize, when you are finished, save it, and pass it as an argument to the `-options-file` flag as shown below:

```
dpacket -source my_sensitive.pcap -options-file my_anon_options.yml
```

## Dpacket Options

The following section describes some more commonly used Dpacket features. Each can be enabled and configured via the Options file or by supplying the equivalent command-line flags.

### Secret Keys

Dpacket includes a simple secret key framework that serves as a basis for encryption routines it performs. It also provides mechanisms for decrypting some encrypted values.

The keys used by Dpacket are randomly generated 128-bit hexadecimal strings. By default, Dpacket automatically generates a new key on each execution. This is the most secure usage mode as no key is reused more than once. If the generated trace file is to be shared with no plan or need for decryption later, this is a recommended selection.

However, if you – the original trace file owner - should ever want to decrypt encrypted values back into their original form, you will need to retain the key. This can become unwieldy if you use many different keys. To address this, Dpacket also accepts a pre-defined secret key using the command-line flag `-key` or via the `key` field in the static Options file. This allows for the reuse of keys across many Dpacket execution runs, ensuring a source IP address is always encrypted to the same value and helps simplify key management.

Finally, to aid in the creation of secret keys, Dpacket provides a command-line flag `-generate-key` that causes it to simply generate a new key and print it to STDOUT as shown below.

```
./dpacket anonymize -generate-key
63eba3a49381930fda89b2ad694e74fe
```

While copying and pasting this value is an option, specifying the `-store-options` flag will cause Dpacket to save a YAML formatted options file containing a dynamically generated key. This is the recommended method if the goal is to produce a static options file, with a static key, for reuse.

### Anonymize MAC Addresses in Ethernet Frame Headers

MAC addresses found in the Ethernet header of an ethernet frame uniquely identify the source and destination interfaces of that frame. Depending on where the traffic sample was captured, the source MAC address may belong to the actual transmitting host, or the nearest upstream router from that capturing device. MAC addresses also contain a 3-byte manufacturer prefix that can be used to identify the manufacturer of the network interface card and possibly the device itself. For these reasons, MAC addresses present significant risk of asset identity or information leakage.

Dpacket provides two MAC address anonymization modes, **full** and **prefix**. In full mode, original MAC address values are replaced with new, randomly generated values that cannot be used to obtain the original value.

In **prefix** mode, dpacket will retain the first 3 bytes (the manufacturer OUI) of the original MAC address and replace the host bytes with a randomly generated value. This is most useful for situations where preserving some information about the device while still obscuring the identity of the host is import to the use-case.

If you'd like to disable full MAC anonymization, use the `-mac ignore` flag or set the `mac_addresses` field in the Options file as shown below:

```
data_link:
  mac_addresses: ignore
```

### Remove or Replace VLAN ID Values in 802.1Q Header

While VLAN ID's do not in themselves pose a privacy exposure risk, but they can be used by would-be attackers to gain an understanding of a network's structure and segmentation. For this, VLAN information presents an elevated risk of unintentional information leakage.

Dpacket offers flexibility in dealing with 802.1Q VLAN headers in the form of a command-line flag `-vlan <mode>`. This command flag allows you to specify one of the following VLAN handling modes:

| | |
|---|---|
| **ignore** | Take no action on VLAN headers (default) |
| **remove** | Remove VLAN headers, update Ethertype in frame |
| **full** | Replace VLAN ID's with pseudorandom number |

When **full** mode is specified, Dpacket randomly selects a new VLAN ID from the range: 2-4093 and replaces the original value with it. This relationship between original and newly selected VLAN ID persists for the remainder of the Dpacket run. dpacket performs recursive anonymization of VLAN tags, ensuring inner, Q-in-Q headers are also anonymized.

When **remove** mode is specified, Dpacket updates the Ethertype of the outer frame header to reflect the type defined in the 802.1Q header, then removes the 802.1Q header entirely. Removing the VLAN header is the recommended mode if a source trace file is known to contain VLAN information that should not be shared. To do this use the `-vlan remove` flag or equivalent Options setting as shown below.

```
data_link:
  vlan_ids: remove
```

### Recalculate Transport and IP Layer Checksums

When modifications are made to an IP header or transport-layer header like UDP or TCP, each header checksum field must be recalculated to reflect the new values. Header values that affect checksum calculation include IP and MAC addresses and since TCP header checksums are calculated using a pseudo-header representing the packet's IP header, changes to the IP header also require the transport header checksum to be recalculated.

Dpacket's default behavior is to recalculate checksums for all modified packets. However, this action can be disabled using the `-fix-checksums false` flag or equivalent Options setting as shown below:

```
fix_checksums: false
```

### Anonymize IP Addresses

IP addresses are among the most important artifacts contained in a PCAP file. They identify the network addresses of devices involved in each recorded communication. IP addresses also pose significant risk of asset and even user identity exposure. For this reason, anonymizing IP addresses effectively is one of Dpacket's core design goals.

For performing full IP address anonymization, Dpacket uses *ipcipher*, a specification for encrypting IPv4 and IPv6 addresses. The ipcipher specification defines reliable, repeatable mechanisms for encrypting IP addresses, specifically, *ipcrypt* for encrypting IPv4 addresses and AES-128 for encrypting IPv6 addresses.

As *ipcipher* uses deterministic algorithms for encrypting IP addresses, using a static key ensures address anonymization done across multiple Dpacket runs, against the same source IP addresses, will produce the same encrypted IP address values. However, under most circumstances, using a randomly generated key is the more secure and preferred approach. If you have no intent of decrypting the IP addresses later, the key can be destroyed. Otherwise, the key should be protected and only shared with stake holders with a specific need to possess it.

Dpacket's IP address anonymization functionality is exposed via the `-ip <mode>` command-line flag and supports the following modes:

| | |
|---|---|
| **full** | Perform full anonymization of all IP addresses found in IP headers (default) |
| **prefix** | Preserve IP prefixes of `-ip-prefix-len` and anonymize remaining host bytes |

| | |
|---|---|
| **map** | Map IP addresses into address space defined in `-ip-mapping-rules` |

When **full** mode is specified, Dpacket encrypts IP addresses found in the IP header of every packet read from the source trace file, this includes IPv4 and IPv6 addresses. This is the default behavior if no IP anonymization mode is specified.

Dpacket makes the **prefix** mode available for preserving IP subnet relationships between hosts. Fully encrypting IP addresses means two addresses belonging to the same subnet, when encrypted, will no longer share a common network prefix. This results in information loss that may hinder some analysis. Using **prefix** mode, you can specify a number of subnet prefix bytes to retain from the original address for IPv4 and IPv6 address separately using the `-ipv4-prefix-len` and `-ipv6-prefix-len` flags.

An alternative to preserving the original network prefix is to replace it with a new one. This allows you to retain the IP subnet relationship between addresses without directly exposing the original network information. Dpacket makes IP **map** mode available for this and when coupled with one or more `-ip-map-rule` flags provides granular control over IP address mappings. IP mapping rules are defined in the format `[original CIDR]=>[target CIDR]` where the target CIDR must contain the same number or more addresses then the original CIDR. Use multiple IP map rules to define mappings for more than one subnet. When dpacket performs an address mapping, it randomly selects an address from the target range and uses that mapping throughout the trace.

To provide more control over what IP addresses dpacket performs anonymization on, dpacket also makes the `-ip-match-mode` and `-match-cidrs` available. These flags allow you to define IP address ranges that should be included or explicitly excluded from IP address anonymization.

### Remove or Anonymize ARP Messages

Address Resolution Protocol (ARP) messages are a fundamental component of Ethernet networks and are used by endpoints and infrastructure devices to resolve an IP address to MAC address and vice versa. ARP messages contain MAC to IP resolution information and can pose a significant information leakage risk and can completely undermine other IP packet anonymization protections applied to a generated trace file.

For this, we recommend excluding ARP messages from trace files to be shared using the BPF filter expression or by specifying the `-arp remove` flag or equivalent Options setting. However, if you choose to retain ARP messages a full anonymization mode is also available:

| | |
|---|---|
| **full** | Perform full anonymization of IP and MAC addresses found in ARP messages |
| **remove** | Do not write ARP messages to the generated trace file |
| **ignore** | Leave ARP messages in place and unchanged (default) |

When **full** mode is selected, Dpacket uses the same IP and MAC address anonymization functions described above to obfuscate addresses found in an ARP message. Specifically, this includes the Sender

and Target Protocol and Hardware address fields. As ARP messages typically contain IP and MAC addresses found in other IP communications, Dpacket ensures the affected addresses are consistently mapped throughout a generated trace file, regardless of the protocol header.

### Remove or Anonymize ICMP Messages

The Internet Control Message Protocol (ICMP) is an IP-layer protocol used for a wide variety of network management and troubleshooting tasks. For example, infrastructure devices such as routers often use ICMP *destination unreachable* or *time exceeded* messages to notify the sender of a packet its intended destination was unreachable. Much like ARP messages, ICMP packets pose significant risk of information leakage as they contain a full IP header as well as a payload. An ICMP payload may contain sensitive data including portions of the sender's original packet.

Like ARP messages, we recommend removing ICMP packets from generated trace files unless they are specifically needed. This can be done using a BPF filter expression, using the `-icmp remove` flag or equivalent Options file setting as shown below.

```
network:
  icmp_messages: ignore
```

If you intend to include ICMP traffic in the generated trace, but wish to obscure the addresses contained with ICMP packets Dpacket also provides a full ICMP packet anonymization mode:

| | |
|---|---|
| **full** | Perform full anonymization of IP addresses found in IP header of ICMP packets |
| **remove** | Do not write ICMP packets to the generated trace file |
| **ignore** | Leave ICMP packets in place and unchanged (default) |

### Packet Truncation to User-Specified Length

One of the major limitations of working with raw PCAP files is their size. To help you address the substantial storage requirements associated with working with raw PCAP files, Dpacket provides the ability to truncate packet payloads that are longer than a specified length.

This option is made available via the `-truncate` flag and the optional `-truncate-len <length>` flag. These options allow you to specify a maximum payload length to which any overly long payload will be truncated. If truncation is enabled but the length is not specified, Dpacket will truncate overly long packet payloads to a default length of 1024 bytes.

While truncating packet payloads can greatly reduce the size of a generated trace file, doing so does have negative side-effects. Often, 3rd-party analysis tools such as Wireshark will annotate affected packets with messages that describe length mismatches and missing or unseen segments. This is expected behavior as it serves as a signal to the researcher that the packets contained in the trace have been truncated.

### BPF Filtering of Packets Read

A very common need – and best practice - when sharing traffic samples is to strip out unwanted traffic. For example, if a researcher has captured all packets on an interface, but they intend to only share HTTP packets, they need an easy way to remove all non-HTTP packets from the trace.

There are many tools that provide this capability when working with raw PCAP files. Dpacket also accommodates this need by allowing you to specify a Berkley Packet Filter (BPF) expression using the `-bpf <filter expression>` command-line flag or equivalent Options setting.

The provided filter is compiled and applied when packets are read from the source trace file, before any anonymization tasks are performed. This reduces Dpacket's overall processing time as well as the size of the generated trace file. We recommend using BPF filters to help ensure only the specific traffic to be shared is included in the generated trace file which greatly helps reduce the risk of accidental information leakage.

### Payload Obfuscation

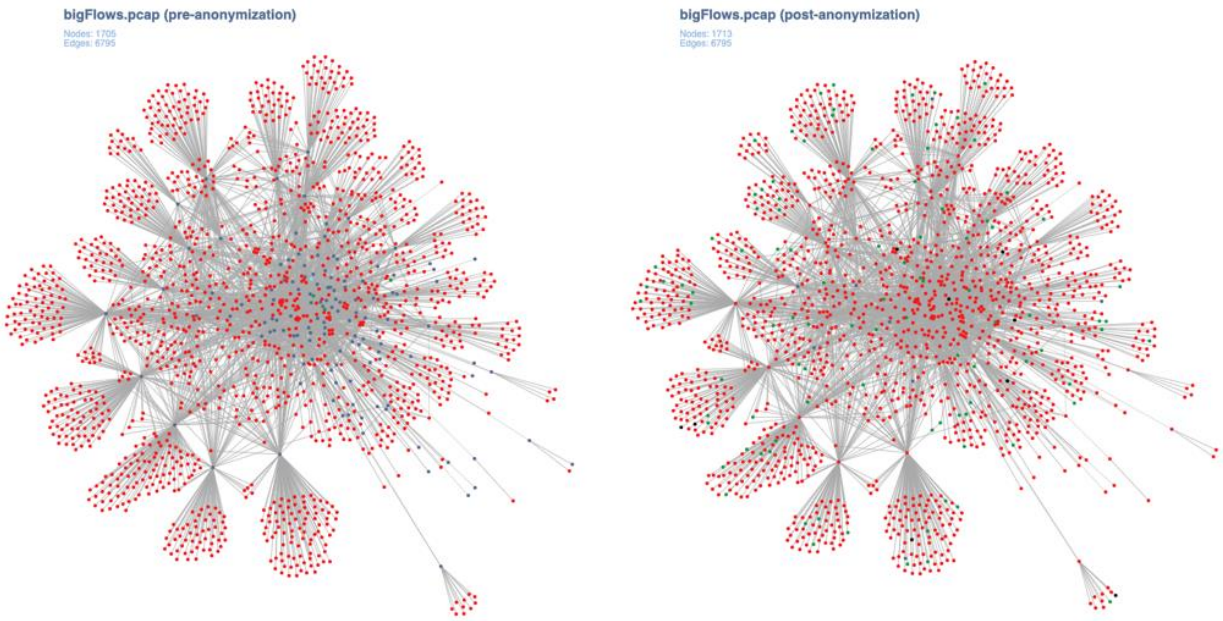
Dpacket implements a basic payload obfuscation capability whereby payload bytes are replaced with a non-sensical value making the original byte value no longer obtainable. Currently the ASCII asterisk character "*" (hexadecimal 2A) is used to obfuscate the actual payload bytes. To enable this feature, specify the `-mask-payloads` command-line flag or equivalent Options file setting as shown below:

```
application:
  mask_payload: true
```

### Network Graph Views of IP Communications

Through this work we have paid particular attention to maintaining entity-to-entity relationships in the generated trace file. To help validate this, Dpacket includes a feature that will cause it to generate network graph visualizations of IP communications contained in the trace file, pre and post anonymization. This feature can be enabled by simply passing the `-graph` command-line flag when Dpacket is executed.

The result is an interactive web page stored in HTML format, in the same directory as the generated trace file. Open the HTML file with a local web browser, and you are presented with a view like that shown below.




The graph on the left represents the IP communications in the trace prior to anonymization. You can zoom in and out, pan the view, and hover over individual nodes to the see their IP address value and highlight adjacent edges and connected nodes.

The graph on the right represents IP communications post-anonymization. The same interactive features are available here, allowing you to explore and validate endpoint relationships as well as IP address mappings.

Additional information is encoded in these graphs in the form of glyph shapes and colors. In summary these encodings are:

- Circle Nodes: IPv4 addresses
- Triangle Nodes: IPv6 addresses
- Green Nodes: Multicast and Unicast addresses
- Black Nodes: Loopback addresses
- Light Blue Nodes: IPv4 broadcast addresses
- Red Nodes: IPv4 non-private addresses
- Dark Blue Nodes: IPv4 RFC 1918 (local) addresses

## Sample Discovery with Dpacket

To anonymize a capture file effectively it is important to know what the file contains. Determining what a capture contains requires manual inspection or analysis with 3rd party tools. Doing this for many files contained in a repository can be a tedious and arduous task. Dpacket now includes an experimental feature that aims to address these issues with the **discover** command.

Dpacket's **discover** command enables an analysis and reporting mode where dpacket will process a file or directory you specify and generate a report detailing many import attributes of each capture file it found. For example, to produce a report describing the capture files in the *pcaps* directory use the command:

```
dpacket discover -dir /pcaps -save-report
```

This causes dpacket to analyze all the readable capture files in the specified directory and generate a JSON-formatted report describing them. The report includes a summary section describing the analyzed data set as a whole and a list of *samples*, each containing information about a specific capture file. For example, the summary and first sample entry from the *pcaps* directory report:


```
{
    "analysis_time": "37.437602642s",
    "total_sample_size": "6.502 Gb",
    "sample_count": 47,
    "earliest": "2016-05-13T13:40:38-04:00",
    "latest": "2023-07-28T15:29:18.001700325-04:00",
    "total_duration": "63169h48m40.001700325s",
    "samples": [
        {
            "name": "2013-08-20_capture-win11.pcap",
            "directory_path": "/Users/rob/pcaps",
            "creation_date": "2022-11-18T12:51:31.360934019-05:00",
            "arp_messages": 23495,
            "data_links": [
                "EN10MB"
            ],
            "capture_duration": "306h5m29.614491s",
            "no_net_layer": 23498,
            "no_transport_layer": 23498,
            "ip_hosts": 1927,
            "icmp_packets": 1703,
            "ipv4_packets": 508871,
            "ipv6_packets": 16571,
            "ip_versions": [
                "ipv6",
                "ipv4"
            ],
            "first_packet_timestamp": "2017-12-31T19:00:50.180622-05:00",
            "last_packet_timestamp": "2017-01-13T13:06:19.795113-05:00",
            "packets": 548940,
            "snaplen": 65535,
            "tcp_packets": 98018,
            "transport_protocols": [
                "udp",
                "tcp"
            ],
            "timestamp_resolution": "10^-6",
            "udp_packets": 425721
        },
…
```


This information can be useful when performing initial triage of a collection of capture samples. It can also provide insights into the type of traffic contained within a sample, in particular attributes that pose an asset identity or information leakage risk.