PktAnon Manual

November 13, 2014

Contents

1

Introduction

PktAnon is a generic framework for anonymization of network traces. The user provides an *anonymization profile* – a small configuration file where he or she describes how individual fields in packets' headers must be processed. An anonymization profile consists of a list of packets, and for each field in the packet header an *anonymization primitive* is specified. Anonymization primitive is a function that transforms an original field into an anonymized field. For example, a user can configure ipv6 addresses to be anonymized using SHA1 anonymization primitive. PktAnon parses the packets from the input source and applies configured anonymization primitives to the listed fields.

The main design decision in PktAnon is that anonymized packets are created from scratch. Fields are copied from the original packets into the new packets after anonymization. Thus no information from the original trace appears in the anonymized trace, unless it is explicitly configured.

PktAnon comes with code to parse common packets and includes several anonymization primitives. PktAnon can also be easily extended with new anonymization primitives and new packets.

1.1 Legend

The following symbols are used throughout this document


warning. important information to consider.


less important information


open issue. it is not known whether and how this feature will be implemented. if implemented, the currently implemented behavior is likely to change


open issue. the feature is not implemented.

The following terminology is used in this document:

packet, record, and protocol data unit in this manual the term *record* is used to refer to the whole packet as it is stored in .pcap file or received by a network card. The terms *packet* and *protocol data unit* are used to refer to a single protocol, such as IP in a Ethernet-IP-UDP packet chain.

SAP SAP stands for *Service Access Point* and means a field in a packet header that is used to select the next-layer protocol (e.g., ethernet `type` field, IP `protocol` field, IPv6 `next header` field)

Anonymization Primitive a function that is used to anonymize fields in a packet. This function accepts field value as an array of bytes and returns another array of bytes that has the same length or is smaller than the original.

1.2 List of Changes from pktanon-1.4

PktAnon-1.0-1.4 were incremental updates. PktAnon-2 is a new implementation of PktAnon with the architecture changed almost completely and most of the code rewritten from scratch. The original ideas of PktAnon are kept in the new code and the list of features is preserved as much as possible. Here we list the changes that are visible to the user.

1. PktAnon now has a new configuration document. We switched from the `triggerconf` format of the previous document to the more descriptive tag names. The example of the new configuration document is shown in Figure 4.1.
2. input and ouput of PktAnon is set using command-line parameters, and not through the configuration file.
3. PktAnon supports new input and output sources in addition to files and system-in/out. For example PktAnon can read packets directly from the network.
4. `libpktanon` includes all code related to packet anonymization and can be easily integrated into other software.
5. current version runs only on linux.
6. PktAnon can now be extended with new packets and anonymization primitives more easily.

1.3 More Information

PktAnon's web site: `http://www.tm.uka.de/software/pktanon/` PktAnon's source can be downloaded from GitHub: `https://github.com/KIT-Telematics/pktanon`

PktAnon is described in the following papers:

- **PktAnon: A Generic Framework for Profile-based Traffic Anonymization**, Thomas Gamer, Christoph P. Mayer and Marcus Schller, PIK Praxis der Informationsverarbeitung und Kommunikation, 2/2008.
- **Datenschutzkonforme Anonymisierung von Datenverkehr auf einem Vermittlungssystem**, Thomas Gamer, Christoph P. Mayer and Marcus Schller, Vortrag auf dem 2. Essener Workshop "'Neue Herausforderungen in der Netzsicherheit"' an der Universitt Duisburg/Essen, October 2006.
- **Datenschutzkonforme Anonymisierung von Datenverkehr auf einem Vermittlungssystem**, Christoph P. Mayer, Bachelor Thesis, Institute of Telematics, University of Karlsruhe, July 2006.

2

Quick Start

2.1 Building PktAnon

PktAnon requires autotools and a compiler that supports C++11. PktAnon uses xml configuration files and hence requires xerces-c[1] xml parser to read the configuration. If you want to use hash functions, then libnettle is also required.

1. download the latest version of PktAnon

```
git clone https://github.com/KIT-Telematics/pktanon.git
```

2. change into the project directory and run the autotools chain

```
cd pktanon
./bootstrap
./configure
./make
```

3. PktAnon executable will be `pktanon` file in the project directory

2.1.1 Running PktAnon

PktAnon need configuration file. Sample configuration files can be found in the `profiles` directory in the distribution. This directory also includes a sample network trace. Two included configuration files are `profile-identity.xml` and `profile.xml`. When used with the former configuration file PktAnon will copy all the fields from original packets into anonymized packets with the exception of packets' content which is cropped. Wth the second configuration file, PktAnon will anonymize mac addresses, ip addresses, and ports in the packets.

`profile.xml` requires PktAnon compiled with libnettle support. libnettle will be used if it can be found on the system.

[1] `xerces.apache.org`

To see available command-line parameters run:

```
$ ./pktanon --help
```

To anonymize sample trace using sample configuration files run:

```
$ ./pktanon -c profiles/profile-identity.xml profiles/sample.pcap out.pcap
```

or

```
$ ./pktanon -c profiles/identity.xml profiles/sample.pcap out.pcap
```

The output should look like this:

```
-----------------------------------------------
pktanon --- profile-based traffic anonymization
-----------------------------------------------
initializing PktAnon,  configuration = profiles/profile-identity.xml
istream: opened file profiles/sample.pcap
ostream: opened output file out.pcap
initialized
complete

statistics for input file 'profiles/sample.pcap'
  processed packets: 9
  errors in packets: 0
  elapsed time:      22us
  Mpps:              0.409
```

The anonymized trace `out.pcap` will be created in the root directory of the distribution.

If the output file already exists it will be trimmed and rewritten without any warning.

3

Building and Running PktAnon

3.1 Dependencies

PktAnon uses autotools.

PktAnon depends on the following software:

required **compiler with C++11 support**. PktAnon requires thread library, functions library, and random number generation library

current version was tested only on gcc 4.8

required **xerces-c**. PktAnon requires xerces to parse its configuration files.

optional **nettle/gnutls**. Anonymization primitives that do cryptographic operations require nettle library. Without the cryptographic library the primitives will not be available.

optional **libpcap**. PktAnon can capture packets with libpcap directly from the network interface.

3.2 Building PktAnon

PktAnon distribution can be downloaded from the PktAnon web-site:

```
git clone https://github.com/KIT-Telematics/pktanon.git
```

Change directory to the root directory of the distribution:

```
cd pktanon
```

Run autotools chain:

```
./bootstrap
./configure
./make
```

Previous versions of PktAnon included anonymization primitives *AnonBytewiseHashSha1* and *AnonBytewiseHashHmacSha1*. They are now deprecated and are not compiled by default. To include them the `--enable-bytewisehashanons` parameter should be passed to the `configure` script. They also depend on cryptographic library.

```
./configure --enable-bytewisehashanons
```

PktAnon executable will be in the root directory of the distribution.

3.3 Running PktAnon

PktAnon requires configuration file as well as configuration of input and output. The list of options can be seen with `pktanon --help`. By default PktAnon will search for file `profile.xml` in the current directory. To use another configuration file use `-c path-to-profile` option.

Following are some examples of how to run PktAnon with different inputs and outputs.

3.3.1 pcap-file to pcap-file

To transform one trace file into another one run call:

```
pktanon -c <anon profile> <input_file> <output_file>
```

This mode is also affected by `-s/--snaplen` parameter: only first snaplen bytes of each packet are read from the file, the rest of the packet is ignored.

If for some reasons libpcap is preferable to std::iostream, use `-l` option to use libpcap for files io.

```
pktanon -c <anon profile> -l <input_file> <output_file>
```

3.3.2 live capture to pcap-file

PktAnon can use libpcap to capture packets from the network. In this case `input_file` parameter is not present, but the `--interface` option must be present.

```
pktanon -c <anon profile> -i <interface> <output_file>
```

Additionally it is possible to specify capture filter (`-e filter`) and not set interface into promiscuous mode (`-p`). Filters are compiled using libpcap. See documentation for libpcap for filter syntax.

```
pktanon -c <anon profile> -i <interface> [-p] [-e <filter>] <output_file>
```

If libpcap is not available, pktanon can use raw sockets (`-r`) to capture files directly. In this case filters are not available.

```
pktanon -c <anon profile> -i <interface> -r [-p]  <output_file>
```

Both libpcap-live and raw sockets require appropriate privileges, most likely running pktanon from superuser.

3.3.3 send anonymized records stream (with pcap header)

PktAnon can send anonymized stream to the remote tcp receiver. Basic syntax is:

```
pktanon -c <anon profile> -o <address>:<port> <input_file>
```

If remote address is IPv6 then add `-6` options. To use udp instead of tcp use `-u`. If tcp is used pcap file header will be sent before any records. If udp is used, pcap file header will not be sent, since it is not possible to know whether it was delivered. Also in case of udp each anonymized packet together with its record header will be sent in separate udp packet.

```
pktanon -c <anon profile> -o <address>:<port> [-6] [-u] <input_file>
```

3.3.4 send anonymized records on the network

PktAnon can send packets directly on the network using libpcap inject functionality.

```
pktanon -c <anon profile> -n <output interface> <input_file>
```

To receive packets from the network and send anonymized packets back on the network use:

```
pktanon -c <anon profile> -i <input interface> -n <output interface>
```

4

PktAnon Configuration File

PktAnon requires a configuration file in xml format. Configuration file speficies how packets should be processed. It consists of global parameters and a so called anonymization profile which says how individual fields in packets should be processed. An example configuration file is shown in Figure 4.1. This section explains general struction of the configuration file and the next two chapters explain supported packets and anonymization primitives.

```
<?xml version="1.0" encoding="UTF-8"?>
<pktanon-config>
  <parameters>
    <param name="recalculate-ipv4-header-checksum">yes</param>
  </parameters>
  <anonymizations>
    <packet protocol="ethernet">
      <field name="mac-source" anon="AnonIdentity" />
      <field name="mac-dest" anon="AnonIdentity" />
      <field name="ethertype" anon="AnonIdentity" />
    </packet>
    <packet protocol="ip">
      <field name="tos"  anon="AnonIdentity"/>
      <field name="identification"  anon="AnonIdentity"/>
      <field name="flags"  anon="AnonIdentity"/>
      <field name="fragment"  anon="AnonIdentity"/>
      <field name="ttl"  anon="AnonIdentity"/>
      <field name="protocol"  anon="AnonIdentity"/>
      <field name="src-ip"  anon="AnonIdentity"/>
      <field name="dest-ip"  anon="AnonIdentity"/>
      <field name="options"  anon="AnonShorten" newlen="0"/>
    </packet>
    <packet protocol="tcp">
      <field name="source-port" anon="AnonIdentity"/>
      <field name="dest-port" anon="AnonIdentity"/>
      <field name="seq"  anon="AnonIdentity"/>
      <field name="ack"  anon="AnonIdentity"/>
      <field name="flags"  anon="AnonIdentity"/>
      <field name="window-size" anon="AnonIdentity"/>
      <field name="urgent-pointer" anon="AnonIdentity"/>
      <field name="options" anon="AnonIdentity"/>
    </packet>
    <packet protocol="payload">
      <field name="payload" anon="AnonShorten" newlen="0"/>
    </packet>
  </anonymizations>
</pktanon-config>
```

Figure 4.1: An Example Configuration File

Parameters in the configuration file can have following values:

boolean parameters parameters which indicate whether the option is enabled or disabled. Values '1', 'yes', and 'on' indicate enabled option and '0', 'no', and 'off' disable the option.

integer parameters must be unsigned integers without delimiters.

bytes must be hexadecimal integers prefixed by '0x', sich as 0xff.

default values some parameters have default values that are indicated by 'auto' or 'default' keyword.

All parameter names and values are *case sensitive*.

4.1 Global Parameters

Global parameters are written in the `parameters` section of the configuration file.

```
<parameters>
  <param name="{param name}">{param value}</param>
</parameters>
```

Currently supported parameters indicate how PktAnon process packets. They are explained in Chapter 5.

4.2 Anonymization Profile

Anonymization profile configures how fields in protocol headers are handled. It includes a list of protocol headers that should be processed and how each field in the header should be anonymized. The list of supported protocols can be found in Chapter 5 and the list of supported anonymization primitives in Chapter 6.

The anonymization profile is enclosed in `<anonymizations>` tags.

```
<anonymizations>
    <!-- anonymization profile -->
</anonymizations>
```

Inside the anonymization profile it is necessary to list all packets whose headers should be anonymized. Configuration of a packet is enclosed in `<packet>` tags and the `protocol` attribute specifies the protocol name.

```
<packet protocol="{protocol name}">
<!-- packet configuration -->
</packet>
```

For each packet it is necessary to list the fields that should appear in the anonymized trace and how these fields must be processed.

```
<packet protocol="{packet name}">
  <field name="{field name}" anon="{anon primitive name}" />
</packet>
```

If the field is not configured in the anonymization profile it will be absent in the output trace (with the exception of recalculated fields, such as lengths and checksums). To copy the field in the anonymization profile without modifying it the field must be present and must use *AnonIdentity* primitive (Section 6.1.4)

If the field must be processed by more than one anonymization primitive, then the list of anonymization primitives can be enclosed in the `<anon>` tag in the `<field>` tag.

```
<packet protocol="{packet name}">
  <field name="{field name}">
    <anon name="{first anon primitive}"/>
    <anon name="{second anon primitive}"/>
  ...
  </field>
</packet>
```

Several anonymization primitives can be useful in, for example, the following situation. It is required to anonymize all mac-addresses of individual computers, but the broadcast addressed must remain broadcast addresses in the trace. In this case one might use an AnonBroadcastHandler, which will prevent anonymization of broadcast addressed by the second primitive, and then the desired primitive for other mac-addresses.

```
<field name="mac-dest">
  <anon name="AnonBroadcastHandler"/>
  <anon name="AnonHashSha1"/>
</field>
```

5

Supported Packets

PktAnon processes traces on a per-record basis. All records are assumed to have the same link-layer protocol. For pcap files the link-layer protocol is determined from the pcap file header. After the link-layer header is processed, PktAnon chooses the apropriate next packet for the next protocol header. This process continues as long as the next header is 1) supported and 2) configured. If PktAnon can't process next header, it calls `PayloadPacket`. The list of supported protocols is shown in Table 5.1.

packet-config	section
ethernet	5.1
linux sll	5.2
vlan	5.3
arp	5.4
ip,ipv4	5.5
ipv6	5.6
icmp	5.7
icmp6	5.8
tcp	5.9
udp	5.10

Table 5.1: Supported Packets

all packets' and fields' names are case-sensitive.

Additionally following parameters affect how packets are processed:

pad-ethernet-packets pad ethernet packets to the minimum lengths of 60 bytes. possible values are `always`, `keep-length` (pad only if original packet length was bigger than 60 bytes), `never` (keep length as it is).

recalculate-ipv4-header-checksum recalculate checksums in ipv4 packets (boolean).

recalculate-payload-checksums recalculate checksums in tcp and udp packets (boolean).

If you are using PktAnon binary, then incomming packets will be trimmed to the specified `snaplen` value (default: 256 bytes) before being processed by the link-layer packet. If you configure PktAnon not to cut packet payloads or some of the packets have long headers than you should change this parameter as well.

5.1 Ethernet

Ethernet packet can be used for ethernet frames with 6 byte MAC addresses. The example configuration for ethernet packet is shown below:

```
<packet protocol="ethernet">
  <field name="mac-source" anon="AnonHashSha1" />
  <field name="mac-dest" anon="AnonHashSha1" />
  <field name="ethertype" anon="AnonIdentity" />
</packet>
```

If the `ethertype` field is not specified, it will be set to 0 in the output packet, possibly making the trace unreadable. It is recommended to use *AnonIdentity* or *AnonKnownEthertype*/ *AnonConstOverwrite* pair for the `ethertype` field.

All fields do not support length changes, that is if the anonymization shrinks the length of the field, the rest of the filled will be filled with nonce.

Based on the `ethertype`, the next packet can be `vlan tag`, `arp`, `ip`, or `ipv6`.

the next packet is selected based on the `ethertype` field before the anonymization.

5.2 Linux Cooked Capture

Linux cooked capture or linux sll is a 'link layer' for captures from *any* network interface. Following fields can be anonymised:

```
<packet protocol="linux_sll">
  <field name="packet-type" anon="AnonIdentity" />
  <field name="ll-address-type" anon="AnonIdentity" />
  <field name="ll-address" anon="AnonIdentity" />
  <field name="protocol" anon="AnonIdentity" />
</packet>
```

The packet is called `linux_sll` with an *underscore*

The next packet is selected based on `protocol` field which is identical to `ethertype` field in `Ethernet` packet.

the next packet is selected based on the `protocol` field before the anonymization.

5.3 Vlan Tag

This packet is not tested.

Following fields of `vlan tag` can be processed:

```
<packet protocol="vlan">
  <field name="priority" anon="AnonIdentity"/>
  <field name="cfi" anon="AnonIdentity"/>
  <field name="id" anon="AnonIdentity"/>
  <field name="ethertype" anon="AnonIdentity"/>
</packet>
```

If the `ethertype` field is not specified, it will be set to 0 in the output packet, possibly making the trace unreadable.

the next packet is selected based on the `ethertype` field before the anonymization.

5.4 ARP

Following fields of ARP packet can be processed:

```
<packet protocol="arp">
  <field name="hardware-type" anon="AnonIdentity"/>
  <field name="protocol-type" anon="AnonIdentity"/>
  <field name="hardware-size" anon="AnonIdentity"/>
  <field name="protocol-size" anon="AnonIdentity"/>
  <field name="opcode" anon="AnonIdentity"/>
  <field name="sender-mac" anon="AnonIdentity"/>
  <field name="sender-ip" anon="AnonIdentity" />
  <field name="target-mac" anon="AnonIdentity"/>
  <field name="target-ip" anon="AnonIdentity"/>
</packet>
```

All fields do not support length changes.

ARP packets are assumed to have IPv4 Address as a protocol address and MAC as a hardware address. The hardware/protocol types and lengthes are not checked.

5.5 IPv4

Following fields of the IPv4 header can be anonymized:

```
<packet protocol="ip">
   <field name="tos"   anon="AnonIdentity"/>
   <field name="identification"anon="AnonIdentity"/>
   <field name="flags"  anon="AnonIdentity"/>
   <field name="ttl"   anon="AnonIdentity"/>
   <field name="protocol"  anon="AnonIdentity"/>
   <field name="src-ip"  anon="AnonIdentity"/>
   <field name="dest-ip"  anon="AnonIdentity"/>
   <field name="options"  anon="AnonShorten" newlen="0"/>
 </packet>
```

In the anonymised packet, the `version` field is set to 4. The fields – `header length` and `header checksum` are recalculated. Checksum recalculation can be turned off.

If the `protocol` field is not specified, it will be set to 0 in the output packet, possibly making the trace unreadable.

If the `fragment offset` is not zero, packet anonymization is stopped at IP header.

packets with options are not tested

IPv4 packet will invoke transformation of IP, UDP, or TCP headers based on `protocol` field.


the next packet is selected based on the `protocol` field before the anonymization.

5.6 IPv6

Following fields of the IPv6 header can be anonymised:

```
<packet protocol="ipv6">
  <field name="traffic-class" anon="AnonIdentity"/>
  <field name="flow-label" anon="AnonIdentity"/>
  <field name="next-header" anon="AnonIdentity"/>
  <field name="hop-limit" anon="AnonIdentity"/>
  <field name="src-ip"  anon="AnonIdentity"/>
  <field name="dest-ip" anon="AnonIdentity"/>
</packet>
```

The `version` field is set to 6. The `payload-length` is recalculated.

IPv6 packet will invoke transformation of IP, UDP, or TCP headers based on `next-header` field.

5.7 ICMP

PktAnon supports anonymization of a generic ICMP header (first 64 bits). The following fields are supported:

```
<packet protocol="icmp">
 <field name="type" anon="AnonIdentity" />
 <field name="code" anon="AnonIdentity" />
 <field name="rest" anon="AnonIdentity" />
</packet>
```

The `rest` is anonymization of bits 32-63 of the ICMP header. The `checksum` field is recalculated. The rest of the ICMP packet is ignored.


processing of ICMP packets is likely to change.

ICMP packet doesn't call processing of further packets.

5.8 ICMPv6 and NDP

PktAnon supports anonymization of a generic ICMPv6 header (first 64 bits). Additionally since PktAnon supports ARP, the support for NDP Neighbor Solicitation and Neighbor Advertisement is included in the ICMPv6 packet processing class. Currently it is impossible to choose whether or not to support NDP messages.

The supported fields for ICMPv6 are:

```
<packet protocol="icmpv6">
  <field name="type" anon="AnonIdentity" />
  <field name="code" anon="AnonIdentity" />
  <field name="rest" anon="AnonIdentity" />
  <field name="target-address" anon="AnonIdentity" />
</packet>
```

The `rest` is anonymization of bits 32-63 of the ICMPv6 header and it is only called if the packet is NOT Neighbor Advertisement or Neighbor Solicitation message. If the message is Neighbor Advertisement or Neighbor Solicitation the value of the `rest` header is set to 0 and the `target-address` field is anonymised as configured. The `checksum` field is recalculated.

Checksums are not recalculated correctly, that is the resulting checksum is not valid.

the R—S—O— flags of Neighbor Advertisement are ignored.

there will most likely be an option on whether to recalculate checksum or not

this transformation is very likely to change

ICMPv6 packet doesn't call processing of further packets.

5.9 TCP

Following TCP fields can be anonymized:

```
<packet protocol="tcp">
  <field name="source-port" anon="AnonIdentity"/>
  <field name="dest-port" anon="AnonIdentity"/>
  <field name="seq"  anon="AnonIdentity"/>
  <field name="ack"  anon="AnonIdentity"/>
  <field name="flags"  anon="AnonIdentity"/>
  <field name="window-size" anon="AnonIdentity"/>
  <field name="urgent-pointer" anon="AnonIdentity"/>
  <field name="options" anon="AnonIdentity"/>
</packet>
```

The `header length` and `checksum` fields are recalculated.

pktanon doesn't check the checksum before processing a packet. Packets with wrong checksums will have their checksum recalculated and set to correct values (unless the anonymization fails for some other reason).

depending on the values of flags, `sequence number` and `ack` fields should have zero values. PktAnon doesn't check the flags and transforms the fields anyway. After the transformation wireshark will show errors in these packets.

packets with options are not tested

After anonymization of TCP header, payload transformation is invoked.

5.10 UDP

Following fields of UDP packet header can be anonymized:

```
<packet protocol="udp">
  <field name="source-port" anon="AnonIdentity"/>
  <field name="dest-port" anon="AnonIdentity"/>
</packet>
```

The `length` and `checksum` fields are recalculated after the anonymization.

pktanon doesn't check the checksum before processing a packet. Packets with wrong checksums will have their checksum recalculated and set to correct values (unless the anonymization fails for some other reason).

After anonymization of UDP header, payload transformation is invoked.

5.11 Payload

`Payload packet` is called when PktAnon can't find any other candidate for the next packet, e.g., unknown ethertypes and protocol values, or `tcp` und `udp` payloads.

```
<packet protocol="payload">
  <field name="payload" anon="AnonShorten" newlen="0"/>
</packet>
```

currently configuring payload packet is required.

PktAnon binary crops incomming records to the value of parameter `snaplen` before processing them. Futher PktAnon was only tested with cropping payloads. Most likely PktAnon will not work with any other anonymization primitive for payload packet.

6

Anonymization primitives

Anonymization primitives are functions that are applied to the values of the fields in packet headers. Function's result is written into the anonymized packet.

PktAnon supports chaining of anonymization primitives, which means that several primitives can be applied to a single field. In this case each primitive will be applied to the result of the previous primitive. Also each primitive can indicate that no more primitives should be applied. This allows, for example, to write so called "filters", which prevent fields with certain values to be anonymized.

PktAnon allows the primitives to reduce the length of the field but not to increase it. Also most header fields "ignore" shortening of the field - that is if the field is shorten, the rest of the bytes will be filled with some random values. This is done so that the resulting traces can be read and interpreted by e.g., tcpdump and wireshark.

PktAnon will translate all numeric packet fields (including ipv4 addresses) to the host byte order before calling the appropriate anonymization primitive and back to the network byte order after the anonymization.

Table 6.1 contains a reference of all included anonymization primitives and their parameters.

All anonymization primitives names and parameter names are case-sensitive.

6.1 Anonymizers

These anonymization primitives change values of anonymized fields. Unless otherwise specified they do not change the lengths of the fields.

6.1.1 AnonConstOverwrite and AnonConstOverwriteRange

AnonConstOverwrite overwrites each byte of the data with a constant byte value. It takes one parameter – `value`, which is byte's value as hexadecimal integer (must preceed with '0x').

```
<field name="mac-dest">
  <anon name="AnonConstOverwrite" value="0x0f"/>
</field>
```

In this example the resulting mac-address field will have a value of `0f-0f-0f-0f-0f-0f`.

AnonConstOverwriteRange allows to overwrite only part of the field. It accepts two additional parameters – `range-begin` and `range-length`.

```
<field name="dest-ip">
  <anon name="AnonConstOverwriteRange" value="0x00" range-begin="2" range-length="2"/>
</field>
```

Name	Parameters	ref	Description
AnonIdentity	-	6.1.4	Preserves original data.
AnonShorten	`newlen`	6.1.7	Cuts the buffer to the given length.
AnonConstOverwrite	`value`	6.1.1	Overwrites every byte with the provided value, given in hex (e. g. 0x00).
AnonRandomize	-	6.1.5	Overwrites each byte with a random value
AnonShuffle	-	6.1.6	Shuffles the bytes of the buffer randomly.
AnonHashSha1	-	6.1.2	Calculates SHA1 hash of the field.
AnonHashSha256	-	6.1.2	Calculates SHA256 hash of the field.
AnonHashHmacSha1	`key`	6.1.3	Calculates HMAC-SHA1 hash of the field with a given key.
AnonBroadcastHandler	-	6.2.1	Preserves broadcast IP or MAC addresses.
AnonKnownEthertypeHandler	-	6.2.2	Preserves ethertypes of protocols that appear in configuration file.
AnonKnownProtocolHandler	-	6.2.3	Preserves protocol numbers that appear in configuration file.
AnonContinuousChar	-		*[not implemented due to difficulty with multithreaded version]* Overwrites every byte with continuous values.
AnonWhitenoise	`strength`		*[very slow]* Applies bit-based white noise of strength between 1 and 10.
AnonCryptoPan	`key`		*[not tested, should work]* Prefix-preserving anonymization. The `key` parameter is needed for the for Rijandel algorithm used inside the prefix-preserving anonymization.
AnonBytewiseHashSha1			*[deprecated and not enabled by default]* Hash every byte separately with SHA1.
AnonBytewiseHashHmacSha1	`key`		*[deprecated and not enabled by default]* Hash every byte separately with HMAC-SHA1. The `key` parameter is needed as key input for the HMAC.

Table 6.1: Overview of anonymization primitives in PktAnon

In this example the IP is trimmed to the first two octets or class B address.

6.1.2 AnonHashSha1 and AnonHashSha256

AnonHashSha1 computes SHA1 value of the field. If the length of the field is smaller than SHA1 output (128 bit = 20 bytes) then the resulting value is truncated to the length of the field (that is fields's value will be the first 'length' bytes of the SHA1 output). If the length of the field is larger than SHA1 output, then the resulting value will be the repeated copies of SHA1 output with the last one possibly truncated (e.g. SHA1Value.SHA1Value.SHA1Va)

AnonHashSha256 computes SHA256 value (which is 256bit = 32 bytes) and is otherwise identical.

6.1.3 AnonHashHmacSha1

AnonHashHmacSha1 computes HMAC hash of the field. It accepts a single parameter – `key` for HMAC. Its behavior is otherwise identical to the *AnonHashSha1*.

6.1.4 AnonIdentity

AnonIdentity copies the field value in the anonymized packet without modifications. Recall that PktAnon doesn't copy any field in the output trace unless it is configured in the configuration file.

```
<!-- in this configuration only mac-source and
ethertype fields will appear in the output trace.
mac-destination field will have undefined value-->
<packet protocol="ethernet">
  <field name="mac-source" anon="{some anon}" />
  <field name="ethertype" anon="{some anon}" />
</packet>
```

```
<!-- in this configuration mac-dest will be copied
from the input trace to the output trace as it is -->
<packet protocol="ethernet">
  <field name="mac-source" anon="{some anon}" />
  <field name="mac-dest" anon="AnonIdentity" />
  <field name="ethertype" anon="{some anon}" />
</packet>
```

6.1.5 AnonRandomize

AnonRandomize replaces the data with a random byte string of the same length.

6.1.6 AnonShuffle

AnonShuffle randomly shuffles the bytes of the field. It applies std::random_shuffle to the provided field treating it as a byte array.

6.1.7 AnonShorten

AnonShorten shortens the length of the field or completely removes it. *AnonShorten* takes one parameter – `newlen`. The new length of the field will be the minimum of its current length and the newlen parameter, e.g., if the length of the field is smaller than newlen the field length will not be changed.

If newlen is zero, field will be removed. This is equivalent to not specifying the anonymization at all.

Most of the packet fields do not support shortening and the shortened fields will be filled with some undefined values. More precisely if, according to the protocol specification, the protocol field must have fixed length its length will be preserved in the output trace. If the field can have variable length it can be shortened and the appropriate length field will be updated. *AnonShorten* can be used on fields like ip or tcp options or packet payloads.

Example:

```
<packet protocol="payload">
  <field name="payload" anon="AnonShorten" newlen="0"/>
</packet>
```

Be very carefull in using *AnonShorten* in chains on fields that don't support length changes. If *AnonShorten* is the first primitive, then PktAnon will copy only newlen number of bytes into the destination. If *AnonShorten* is not the first primitive in chain, than all primitives before will

proceed, length will be shortened without overwriting rest of the values and when the field ignores lengh change all bytes after newlen will also appear.

6.2 Filters

These primitives can be used to prevent certain values of fields from anonymization. They must appear in the configuration before the anonymizer.

6.2.1 AnonBroadcastHandler

This primitive prevents anonymization of broadcast IP and MAC addresses. Addresses, that have all bits set to 1 are considered broadcast addresses. It must be included before any other primitive to process the address field. It doesn't have any parameters:

```
<field name="mac-dest">
  <anon name="AnonBroadcastHandler"/>
  <anon name="{anon for the field}"/>
</field>
```

6.2.2 AnonKnownEthertypeHandler

AnonKnownEthertypeHandler can be used to filter out ethertypes of the packets that are present in the configuration. It can be used to anonymize ethertypes of "unknown" packets.

AnonKnownEthertypeHandler can be applied to Ethernet::ethertype field.

6.2.3 AnonKnownProtocolHandler

AnonKnownProtocolHandler does the same as AnonKnownEthertypeHandler but applies to IP::protocol and IPv6::next_header fiedls.