Pcap randomization tool documentation

How to run the program

The program is a console application. It is written in Python 3, therefore it needs Python interpreter. The Scapy, NumPy and configparser libraries are also required. There are multiple ways of running the program from console:

- python diplomka.py <name of pcap file> – randomizes the pcap file using config.ini file (default name).
- python diplomka.py <name of pcap file> [name of config file] – randomizes the pcap file using given config file.
- python diplomka.py -g – generates default config file named config.ini.
- python diplomka.py -g [name of config file] – generates a new config file with given name.

Name of the output file can be changed in the config file, default is out.pcap. The config file is vital for the program. It contains all settings of randomization. User needs to edit the config file and then run the program afterwards in order to get desired outcome. Please do not change the structure of the program folder.

Config file description

Config file (with default name config.ini) influences the outcome of randomization and is similar to Windows ini files. It consists of a number of named sections. Each section then consists of key=value pairs. Values must fit these conditions:

- Logical values must be written as true or false.
- Decimal numbers are written normally, in case of real numbers with decimal point (for example 10 or 3.14)
- Text values are entered in special situations and will be described later.

The first section is named “ALL” and contains general settings:

- changeindividualpackets: decides if a per-packet or per-stream processing should be used.
  - Per-packet processing will randomize every packet independently and all relations between packets will be lost.
  - Per-stream processing will group packets into streams based on IP addresses, ports and in case of RTP and RTCP even SSRC identifier. Each stream will be randomized independently and packets within the same stream will share IP addresses, ports, capture time shifts and SSRC identifiers. Everything else will be randomized on per-packet basis.
- stream_processing_version: decides version of per-stream processing.

    - s1: most simple version – Only packets with same source and destination IP addresses and ports will be grouped into the same stream. Incomming and outgoing communications will be randomized differently.
    - s2: advanced version – Incomming and outgoing communications will be grouped together into one stream but if packets have same IP addresses and different ports, they will not be grouped.
    - s3: maximal version – like s2, but packets with same IP address will have same IP address after randomization even if they have different ports. But communication between three or more IP addresses where one is shared will still not be grouped together.
- delete_stream_chance: Chance to delete stream in percents (0-100). In per-packet processing is used, this field is ignored.
- duplicate_stream_chance: Chance to duplicate stream in percents (0-100). In per-packet processing is used, this field is ignored.
- stream_duplication_version: Allowed values are 1 or 2.
    - version 1: stream is duplicated first and then randomized. Duplicated streams will be different after randomization.
    - Version 2: stream is randomized first and then duplicated. Duplicated streams will be the same.
- output_pcap_file: Name of the result pcap file.
- minimum_rtp_packets_in_stream: Minimal number of potential RTP and RTCP packets in a stream needed to confirm their allegiance.

After the “ALL” section there is a section for every supported protocol. Each protocol can be randomized differently. All sections contain these settings:

- delete_packet_chance: Chance to delete packet in percents (per-packet processing).
- duplicate_packet_chance: Chance to duplicate packet in percents (per-packet processing).
- duplication _version: Version of duplication.
- source_ip_change_chance and destination_ip_change_chance: Chance to randomize source and destination IP addresses.
- ipv4_randomize_parts: Which part of the IP address should be randomized. Values:
    - network: Only network part of the IP address will be randomized.
    - host: Only host part of the address will be randomized.
    - both: The entire IP address will be randomized.
- ipv4_mask_length: Number of bits of the network part of IP addresses. These fields are the same for ipv6.
- source_port_chance_change and destination_port_chance_change: Chance to change ports.
- source_port_min and source_port_max: These keys set boundaries of interval, from which new values for ports will be randomized.
- time_chance_change: Chance to change capture time of packets.
- time_randomization_method: Values:
    - normal: Number will be generated from normal distribution. Time_constant means standard deviation.

    - direct: Packet capture time will be modified by adding the given value in seconds (time_constant). Can be negative.
- time_constant: Has different meaning based on time_randomization_method. It is a real number.

After that there are settings that are specific for each protocol. These settings modify randomization of different header fields. Each field always has a chance to randomize value and then randomization interval boundaries. These fields are easy to understand and they are described in the config file. Minimal and maximal value for intervals are also in config file.

Folder structure

Folder contains:
- all python files
- config.ini file with configuration
- out.pcap file with results of randomization
- documentation.pdf which is this file
- test_pcap folder with pcap files for testing
- config_files with example configuration files

Module and function documentation

Program consists of 12 python files. The main one is called diplomka.py.

diplomka.py:

Main file of this script. Controls computation flow. Important functions:
- parse_packets: Main function of this program. Loads everything, then runs packet classification, then randomizes packets and finally sorts and writes packets to pcap file. Takes no arguments and returns nothing.

decoder.py:

This file contains the Decoder class. The main goal of a Decoder is to decode packets - use pattern matching on the highest layer (application layer) to find out what protocol is there. For every protocol that should be recognized by this script there must be a method which will implement pattern matching on packet data and will be able to decide if the packet's protocol corresponds to the method. Each method takes a packet that should be decoded and returns either True (protocol matched) or False (protocol does not match).

Contains methods for packet classification: isRTCPpacket, isRTPpacket, isSTUNpacket, isTLSpacket, isDTLSpacket. Each method takes a packet and decides if the packet belongs to corresponding protocol. Returns true if yes and false if no.

packet_classes.py:

This file contains MyPacket class as well as RTP_packet, RTCP_packet, STUN_packet, TLS_packet, DTLS_packet that inherit from MyPacket class. These classes wrap scapy packet representation and add additional functionality. Every class mainly contains functions for randomization of itself.

MyPacket serves as a parent class for the others. It contains functions for recalculating checksums, converting packets to scapy representation for writing, getters and setters. Other classes contain mainly the randomize function. They take information about per-stream or per-packet processing. Then they conditionally take values for individual fields of header. If they are set as None, fields will be randomized, if they have value the value will be assigned to packet and not randomized. Setting values directly is used for per-stream processing. Other important functions are setters, that set individual fields in headers. Packet is represented as a list of bytes so assigning values is a little bit complicated.

packet_classification.py:

This file implements the classification part of this program. It contains one function classification_function that goes through a list of all packets and tries to classify them by checking their type and then assigning them a class. It also inserts packets into streams if per-stream processing is used.

RTP_stream.py:

This class contains RTP streams. RTP stream includes RTP and RTCP packets. RTP streams are stored in a dictionary where key is a stream identificator and value is a list of packets that belong to the stream. Stream identificator is a tuple which contains source and destination IP, source and destination port and SSRC. Streams that are too short can be removed using method stream_check. Streams are added using method add.

Formally: streams = {(src_ip, dst_ip, src_port, dst_port, ssrc): [MyPacket, …]}

per_packet.py:

This file implements the per-packet processing. There is only one function process_packets. The function goes through all packets in the list and randomizes them. It expects a list of packets and returns a list of randomized packets. It is relatively simple, only complexity is added by packet removal and duplication.

per_stream.py:

This class further divides packets into substreams based on their protocol after they have been divided by IP and port addresses. Each protocol can also be divided even more, for example RTP/RTCP (using SSRC). This class also implements the per-stream randomization of all packets.

Method add adds new packets to the stream. Method generate_conversion_dict makes assigning same values to multiple packets in one stream possible. In the beginning of stream processing this dictionary is generate and contains rules of translations for header values. Protocol_based_randomization and process_packets_in_stream are experimental methods and are not used in the final program.

Then there are methods for processing individual streams based on protocols. Each method goes through all packets in stream and randomizes them. These methods are run from process_streams which helps to deal with removal and duplication of streams. Methods remove_streams and duplicate_streams go through all streams and remove or duplicate them.

In the end there is method get_packets which is used to extract all packets from the class in order to write them into pcap file after randomization. It returns a list of packets.

Following three classes are used to divide packets into streams based on IP and ports without considering different protocols or anything else. These classes perform basic distribution of packets into streams and then pass these packets into the per-stream class for more delicate distribution. They generally contain method for adding packets, processing packets (which is done by per_stream class) and extracting packets from streams.

s1.py:

This class is used for per-stream processing. It is the most basic one - only packets with exactly the same source, destination IP address and source, destination port will be put together. Therefore only packets from PC1 to PC2 on the same ports will have the same IP and port information after randomization. Packets from PC2 to PC1 will have different IP and port.

The class contains a dictionary where key is a stream identificator and value is a list of packets that belong to the stream. Stream identificator is a tuple which contains source and destination IP, source and destination port. Formally: self.streams = {(src_ip, dst_ip, src_port, dst_port): [Packet1, …]}

s2.py:

This class is used for per-stream processing. It is the advanced one – packets with similar source, destination IP address and source, destination port will be put together. Similar means that source and destination IP or ports can be switched. Therefore both packets from PC1 to PC2 and from PC2 to PC1 using the same ports will have the same IP and port information after randomization. Communication between same PCs but on two or more sets of ports will result into different IP and port after randomization.

The class contains a dictionary where key is a stream identificator and value is a list of packets that belong to the stream. Stream identificator is a tuple which contains source and destination IP, source and destination port. The difference between this class and S1 is in

extended check whether the packet belongs to the stream but structurally is the same. Formally: self.streams = {(src_ip, dst_ip, src_port, dst_port): [Packet1, …]}

s3.py:

This class is used for per-stream processing. It is the elite one – packets with similar source, destination IP address and source, destination port will be put together. Similar means that source and destination IP or ports can be switched. Therefore both packets from PC1 to PC2 and from PC2 to PC1 using the same ports will have the same IP and port information after randomization. Communication between same PCs but on two or more sets of ports will also be included and will have same IP addresses after randomization. Ports will be different.

The class contains a dictionary where key is an IP stream identificator and value is a dictionary of port stream identificators. Value of the second dictionary is a list of packets that belong to the stream. IP stream identificator is a tuple which contains source and destination IP. Port stream identificator is a tuple which contains source and destination port. Formally: self.streams = {(src_ip, dst_ip): {(src_port, dst_port): [MyPacket, …]}}

utility.py:

This file contains smaller useful functions that help with computations like IP address generation or packet data conversion between scapy format and the format used in this program. These functions are well documented in the file.

config_parser.py:

Config file parser. This module is responsible for loading, parsing and creating config file. Firstly it loads the config file using parse_config_file. Secondly it can generate a default config file using generate_config_file, thirdly there are functions used for checking correctness of loaded values from config file. These functions are divided into smaller ones in order to preserve some level of readability. There are lots of settings in the config file that need to be checked.