CCgen (Covert Channel generator)

- *. . . 2020, FM*
- *Nov 2021, FIV*

Introduction

CCgen is a tool for creating, transmitting, and retrieving covert channels into TCP/IP features and packets.

CCgen has two operation modi: (a) **online**, in which covert channels are generated on-the-fly and transmitted to an aimed destination address, and (b) **offline**, in which covert channels are injected into existing flows of pcap captures. It also shows two operation roles: (1) **injector**, which creates and/or injects a covert channel, and (2) **extractor**, which receives and/or discloses a covert channel.

Hence, four different operational behaviours are possible: (a1) online injector, (a2) online extractor, (b1) offline injector, (b2) offline extractor.

CCgen operational modi

Depending on the desired operational behaviour, CCgen accepts the following calls from a terminal:

- Online injector:

  ```
  $ python3 ccgen.py online inject <configuration_file>
  ```

- Online extractor:

  ```
  $ python3 ccgen.py online extract <configuration_file>
  ```

- Offline injector:

  ```
  $ python3 ccgen.py offline inject <configuration_file>
  ```

- Offline extractor:

  ```
  $ python3 ccgen.py offline extract <configuration_file>
  ```

Configuration file

The configuration file is a plain TEXT file divided into three blocks:

- [Files] contains information about the files to use as inputs and outputs for the injection or extraction of the covert channel. It includes the following fields:
  - *input:* the original pcap to read.
  - *output:* the pcap generated by CCgen with the injected covert channel.
  - *message:* the path to the binary message text file

  - *mapping:* the path to the mapping file. This file contains parameters and the relationship between the message and the field in which the covert channel is created. The *docs/techniques.md* links techniques with suitable mapping files examples included in the *MappingFiles* directory.
- [Filter]
  - *src_ip:* source IP address established as covert channel sender.
  - *dst_ip:* destination IP address established as covert channel receiver.
  - *src_port:* source port that the sender uses to send the covert channel.
  - *dst_port:* destination port in which the receiver receives the covert channel.
  - *proto:* protocol used by the sender and receiver of the covert channel
- [Iptables] is only required in the online modi
  - *chain:* ...
  - *queue:* ...
- [Channel]
  - *technique:* technique that will be used for injecting. Implemented techniques are documented in the *docs/techniques.md* file and have a corresponding python file in the [techniques] folder.
  - *bits:* slice the message into x bits (this should be compatible with the mapping used).
  - *layer:* layer where the covert channel is injected (IP, TCP or UDP). Generally IP should work in all cases.

(*Note that, depending on the technique and flowkey used, some of these fields/parameters are not necessary.*)

Examples of configuration files Offline injection:

```
[Files]
input: in.pcap
output: out.pcap
message: message_to_inject.txt
mapping: MappingFiles/mapping_bin.csv

[Filter]
src_ip: 200.60.153.149
dst_ip: 15.161.162.34

[Channel]
technique: ipflags
bits: 1
layer: IP
```

Offline extraction:

```
[Files]
input: out.pcap
message: message_extracted.txt
mapping: MappingFiles/mapping_bin.csv

[Filter]
src_ip: 200.60.153.149
dst_ip: 15.161.162.34

[Channel]
technique: ipflags
bits: 1
layer: IP
```

Online injection:

```
[Files]
message: message_to_inject.txt
mapping: MappingFiles/mapping_bin.csv

[Filter]
src_ip: 10.0.0.1
dst_ip: 10.0.0.2

[Iptables]
chain = OUTPUT
queue = 2

[Channel]
technique: ipflags
bits: 1
layer: IP
```

Online extraction:

```
[Files]
message: message_extracted.txt
mapping: MappingFiles/mapping_bin.csv
```

```
[Filter]
src_ip: 10.0.0.1
dst_ip: 10.0.0.2

[Iptables]
chain = INPUT
queue = 1

[Channel]
technique: ipflags
bits: 1
layer: IP
```