py__CCgen

Generate, inject and extract covert channels in network traffic

FIV, Nov 2021

Introduction

This repository includes:

- The **CCgen** tool for generating, injecting and extracting covert channels in network traffic
- A **wrapper for CCgen**. The wrapper automatically searches for matching flows in a given pcap and generates suitable configuration files for injecting multiples covert channels with CCgen.
- **Related tools** for transforming text and binary data as well as for extracting flows with different flow-keys with go-flows.

Folders

- ccgen includes the *CCgen* tool.

- [pcaps] contains a small *test.pcap* traffic capture to test the tool and run simple examples. This pcap is a portion of the captures provided by the MAWI Working Group Traffic Archive in https://mawi.wide.ad.jp/mawi/
- [txts] contains *hello.txt*, an example of a short text file to be injected in covert channels.
- [txts_bincoded] contains *hello_bin.txt*, which is the binary codification of *hello.txt*
- [spammer] contains several files on how to configure an infrastructure to run CCgen in the online modus.
- [utils] includes tools for transforming data and generating flow extraction configuration files.
- [wrapper] contains and saves relevant files related to the wrapper.

CCgen

Read the *README.md* file within the ccgen folder for comprehensive information, also the documentation in the [ccgen/docs] folder.

Running the CCgen-wrapper

The main purpose of the CCgen-wrapper is to allow the automatic injection of multiple covert channels in the same pcap. It has three different parts:

1. Searching for matching flows and creating corresponding ccGen configuration files.
2. Injecting the covert channels.

3. Extracting the covert channels to evaluate the previous injection

To run the wrapper follow the steps below:

1. Download and install go-flows The CCgen wrapper requires installing go-flows within the [wrapper] folder. Download **go-flows** from: https://github.com/CN-TU/go-flows and make sure that the folder is named [go-flows-master].

2. Create a configuration file for the CCgen-wrapper The CCgen wrapper is called with a wrapper configuration file. You can find a default example named **config.wrp** in this folder. This configuration file has a CSV-table format with a header in which each row stands for a covert channel configuration. The features (or columns) of the configuration are:

- *message_file*: the file with the message to covertly send. It must be a text file with a sequence of '0s' and '1s'.
- *technique*: the technique to use to create the covert channel. You can find a list of implemented techniques with descriptions in the *ccgen/docs/techniques.md* file.
- *key*: stands for the flowkey to inject the channel. Options are: "1tup" (for srcIP), "2tup" (for -srcIP, dstIP-), "3tup" (for -srcIP, dstIP, Protocol-), "4tup" (for -srcIP, dstIP, srcPort, dstPort-), and "5tup" (for -srcIP, dstIP, Protocol, srcPort, dstPort-)
- *mapping*: refers to the parameters and symbol-to-value correspondence to use in the injection of the covert channels. In the*ccgen/docs/techniques.md* file, together with the description of the *technique*, you will find suitable examples of mapping files, which are stored in the *ccgen/MappingFiles* folder.
- *bitspkt*: accounts for the number of covert bits that each packet (or transition between packets) contains.
- *const*: is included to specify additional constraints. This is important for techniques that can only be implemented in certain protocols. Implemented options: "None" (default), "tcp", "udp", "tcp/udp".
- *rep*: is the number of repetition that the same configuration must be injected in different flows (by default '1').

Important! Note that wrong, misleading configurations are possible. In such cases, the wrapper will not check any consistency and the whole process will simply fail at some point.

3. Create CCgen configuration files Before injecting cover channels, configuration files for ccGen must be generated. To do this, run:

```
> python3 genCCconfigs.py <wrapper_config_file> <input_pcap> <output_pcap>
```

where the *wrapper_config_file* is the file described in the previous Step 2, the *input_pcap* is the original traffic capture in pcap format and the *output_pcap* is the name of the resulting pcap containing the covert channels. You can run the default example:

```
> python3 genCCconfigs.py config.wrp pcaps/test.pcap pcaps/out.pcap
```

Note that **genCCconfigs.py** uses and creates files within the [wrapper] folder. Configuration files for the injection with ccGen are saved in the [wrapper/ccgen_inj_config] folder, and the corresponding extraction in the [wrapper/ccgen_ext_config] folder.

4. Inject covert channels with CCgen-wrapper To inject cover channels in a pcap, run:

```
> python3 ccgen_wrapper_inj.py <ccgen_inj_config_folder>
```

Where *ccgen_inj_config_folder* is the folder with the ccGen configuration files for injection previously created with *genCCconfigs.py. A default example can be run with:

```
> python3 ccgen_wrapper_inj.py wrapper/ccgen_inj_config/
```

5. Extract covert channels with CCgen-wrapper for evaluation To check if injected cover channels have been correctly injected in the pcap, run:

```
> python3 ccgen_wrapper_inj.py <ccgen_ext_config_folder>
```

Where *ccgen_ext_config_folder* is the folder with the ccGen configuration files for extraction previously created with *genCCconfigs.py. A default example can be run with:

```
> python3 ccgen_wrapper_ext.py wrapper/ccgen_ext_config/
```

Extracted messages are saved as text files in this folder. *ccgen_wrapper_ext.py* shows them automatically at the end of the complete extraction process. However, you can use *bin2text* to decode them individually.

Text2bin and bin2text

text2bin.py and *bin2text.py* script are quite simple and straightforward. To see how they work, simply run:

```
> python3 utils/text2bin.py txts/hello.txt
```

```
> python3 utils/bin2text.py txts_bincoded/hello_bin.txt
```

Running CCgen in Online-Mode

To inject Covert Channels on the fly, refer to the spammer folder where we give a detailed explanation with an easy setup.

Acknowledgments

This work was supported by the project MALware cOmmunication in cRitical Infrastructures (MALORI), funded by the Austrian security research programme KIRAS of the Federal Ministry for Agriculture, Regions and Tourism (BMLRT) under grant no. 873511.