A Brief CCEAP Documentation

v. 0.1.2 by Dr. Steffen Wendzel

(**Note:** An initial academic publication is currently under review for a conference; it provides more information on CCEAP and will be linked here in case of acceptance and publication.)

The *Covert Channel Educational Analysis Protocol* (CCEAP) is a network protocol designed for teaching covert channels to professionals and students. For an introduction on network covert channels (network steganography), please have a look at this publication.

The protocol is explicitly vulnerable against several hiding patterns so that switching protocols while explaining hiding patterns is not necessary. The protocol's structure is simple and self-explanatory and its implementation is kept at a minimum level of code lines to make it especially accessible to students.

Please send requests and feedback to the author: Steffen Wendzel, www.wendzel.de (steffen (at) wendzel (dot) de).

Research on covert channel teaching is currently performed by Steffen Wendzel and Wojciech Mazurczyk.

Installation

The tool requires gcc and Linux (or similar). Run `make` to build the two components of the tool: **client** and **server**.

Teaching Process in a Nutshell

The educational process is split into two parts. First, fundamentals are explained, secondly, exercises are performed and evaluated.

1. Preparing Students to Use CCEAP The lecturer has to introduce - the CCEAP protocol (see below), and - Hiding patterns as they are described on our network information hiding patterns website that also points to several of our recent publications in which we introduce the topic.

2. Exercises In the exercises, students should try to create covert channels **or** should try to determine the type (pattern) of a given covert channel. Therefore, the lecturer can perform one of two exercises:

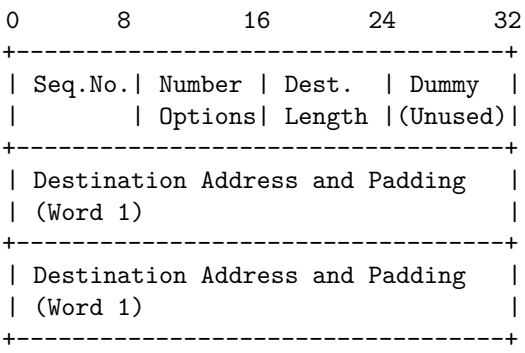
- The lecturer can ask the *students to establish a hidden communication that represents a given pattern.* The pattern can be one of the known patterns of

the available pattern collection, e.g. ValueModulation or Reserved/Unused. Therefore, the students need to find out how to create a covert channel for the particular pattern using the CCEAP protocol by analyzing the protocol structure and the `client` tool's command line parameters.
- Alternatively, the lecturer can ask the students to *determine the hiding pattern for a given traffic recording or for given tool parameters* that were used to run the client.

CCEAP Protocol

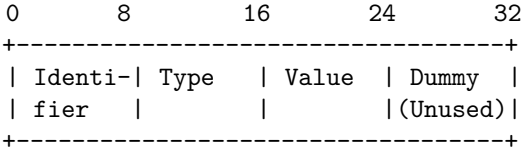
The protocol contains two components: A CCEAP main header and options headers (between 0-128). The options headers will be concatenated to the main header.

Main Header The main header contains three words. The first contains a *Sequence Number* (8 bit), the indicator of how many options headers are attached (*Number of Options*, 8 bits), the length of the destination address in bytes (*Destination Length*, 8 bits) and an unused *Dummy* field (8 bits). The following two words contain the destination address (an ASCII value) that is padded with `X` bytes if too short for two words.



The sequence number is incremental and starts with 1 by default (this can all be configured!). The number of options is zero by default.

Options Headers The options header contains a freely definable *Identifier*, *Type* and *Value* field (8 bits each) and another *Dummy* field (also 8 bit).



Tool Architecture and User-Guide

The implementation of the CCEAP protocol is split into two components, a client and a server. The client transfers CCEAP protocol packets to the server. Users can freely define how the particular fields of the CCEAP header are filled and how optional header components are embedded. The server, on the other hand, simply displays the received content which allows to view whether packet data were transmitted in the desired way.

All actions can be performed over the network or on the local host. The easiest way is to open two X terminals (e.g. GNOME Terminal) and start the server in one terminal and the client in the other.

Example Walk-through for an Exercise Let us assume that the exercise given to the students was to create a covert channel that uses the Reserved/Unused pattern. The students look up the pattern description, which tells:

The covert channel encodes hidden data into a reserved or unused header/PDU element.

By analyzing the protocol structure of CCEAP, the students find an unused header field, namely the *Dummy* field, which is 8 bits in size. The field could be used to place hidden data in the way as described by the pattern. By running `./client -h`, the students check whether they can manipulate the Dummy field in some way:

```
$ ./client -h
CCEAP protocol implementation. Copyright (C) 2016 Steffen Wendzel
...
The following parameters are supported by CCEAP client v.0.4.1:
...
-D x   Destination IP x to connect to
-P x   TCP port x to connect to
...
-u x   Use digit x instead of 0 as `dummy' value in the main header
```

The parameter `-u` can obviously be used for creating a Reserved/Unused pattern-based covert channel. To this end, the students run the server on the localhost port 2222:

```
$ ./server -P 2222
...
CCEAP - Covert Channel Educational Analysis Protocol (Server)
   => version: 0.4.1, written by: Steffen Wendzel, www.wendzel.de
```

Please note that the use of an additional `-v` parameter provides verbose output for the client and for the server and that the parameter `-h` works for both, `server` and `client`.

Next, the students run the client and connect to localhost (127.0.0.1), port 2222. They set the value of the Dummy field to 42:

```
$ ./client -D 127.0.0.1 -P 2222 -u 42
...
connecting ... connected.
sending: ..........
sending done.
```

By default, the client sends 10 packets with incremental sequence number. This is why the server will receive and display now 10 packets with the same content. Of course, such parameters can be changed as desired. The typical packet output is shown as follows:

```
received data (12 bytes):
 > sequence number:       1
 > destination length:    0
 > dummy value:           42
 > destination + padding: XXXXXXXX
 > number of options:     0
```

More complex outputs will be provided when more complex packets are received by the server, e.g. packets containing optional headers.