CCHEF – Covert Channels Evaluation Framework Design and Implementation

Sebastian Zander, Grenville Armitage
Centre for Advanced Internet Architectures, Technical Report 080530A
Swinburne University of Technology
Melbourne, Australia
szander@swin.edu.au, garmitage@swin.edu.au

***Abstract*—Communication is not necessarily made secure by the use of encryption alone. The mere existence of communication is often enough to raise suspicion and trigger investigative actions. Covert channels aim to hide the very existence of the communication. The huge amount of data and vast number of different protocols in the Internet makes it ideal as a high-bandwidth vehicle for covert communications. Covert channels are hidden inside pre-existing overt communication by encoding additional semantics onto 'normal' behaviours of the overt channels. We have developed CCHEF – a flexible and extensible software framework for evaluating covert channels in network protocols. The framework is able to establish covert channels across real networks using real overt traffic, but can also emulate covert channels based on overt traffic previously collected in trace files. In this paper we present the design and implementation of CCHEF.**

***Index Terms*—Covert Channels, Network Protocols**

I. Introduction

Often it is thought that the use of encryption is sufficient to secure communication. However, encryption only prevents unauthorised parties from decoding the communication. In many cases the simple existence of communication or changes in communication patterns, such as an increased message frequency, are enough to raise suspicion and reveal the onset of events. Covert channels aim to hide the very existence of the communication. They hide within pre-existing (overt) communications channels by encoding additional semantics onto 'normal' behaviours of the overt channels.

Lampson introduced covert channels as a means to secretly leak information between different processes on monolithic systems [1]. In recent years the focus has shifted to covert channels in network protocols [2], [3]. The huge amount of data and vast number of different protocols in the Internet makes it ideal as a high-bandwidth vehicle for covert communications. The capacity of covert channels in computer networks has greatly increased because of new high-speed network technologies, and this trend is likely to continue. Even if only one bit per packet can be covertly transmitted, a large Internet site could lose 26 GB of data annually [4].

Covert channels are primarily used to circumvent existing information security policies, to exfiltrate information from an organisation or country in a manner that does not raise suspicions of the network owners or operators. Although network covert channels may not be used frequently today, because of increased measures against 'open channels', such as the free transfer of memory sticks in and out of organisations, the use of covert channels in computer networks will increase in the near future [5].

We have developed a software framework to evaluate covert channels in network protocols called Covert Channels Evaluation Framework (CCHEF). The main goals of CCHEF are to evaluate the capacity, stealth and robustness of network covert channels. Since CCHEF creates network covert channels, it also can provide traffic data for evaluating countermeasures against network covert channels – mechanisms for their elimination, capacity limitation or detection. Furthermore, CCHEF is able to create traffic for testing existing firewalls or intrusion detection software.

CCHEF can be used in real networks with real overt traffic, but can also emulate covert channels using overt traffic from trace files. Usually testing with real traffic is restricted to controlled testbeds where it is almost impossible to generate a realistic traffic mix from a larger number of users/hosts. Therefore, CCHEF also runs on single hosts emulating covert channels based on overt traffic from trace files. CCHEF supports both covert channels hidden in data fields (storage channels) and the timing of packets (timing channels).

The architecture of CCHEF is very flexible and extensible. New covert channels modules can be added without the need to modify the framework itself. Furthermore, new mechanisms for covert channel security (authentication, encryption), framing or (reliable) transport can easily be added to CCHEF.

The paper is organised as follows. Section III describe the overall goals and main features of CCHEF. Section IV describes the architecture of CCHEF. Section V provides an overview how CCHEF can be configured to perform different tasks. A more detailed description of how to configure and use CCHEF is provided in the user manual [6]. Section VI concludes and discusses future work.

II. Covert Channels Overview

A. Covert Communication

The de-facto standard model for covert channel communication is the *prisoner problem* posed by Simmons [7]. Two people, Alice and Bob, are thrown into prison and intend to escape. To agree on an escape plan they need to communicate, but all their messages are monitored by Wendy (the warden). If Wendy finds any signs of suspicious messages she will place Alice and Bob into solitary confinement preventing any escape. Alice and Bob must exchange innocuous messages containing hidden information that (ideally) Wendy will not notice.

Extending this scenario towards communication networks, Alice and Bob use two networked computers to communicate. They run some innocuous *overt* communication between their computers, with a hidden *covert* channel inside. Alice and Bob share a secret useful for determining covert channel encoding parameters and for encrypting/authenticating the hidden messages. For practical purposes Alice and Bob may well be the same person (e.g. a hacker ex-filtrating restricted information). Wendy manages the network and can monitor the passing traffic for covert channels or alter the passing traffic to disrupt or eliminate covert channels. Figure 1 depicts the communication model (Alice sending to Bob).

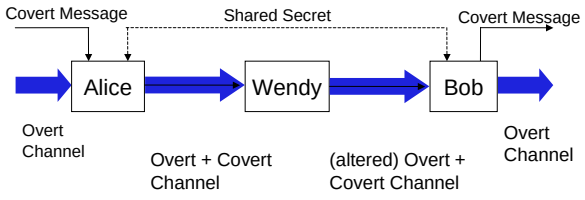

Figure 1. The prisoner problem – model for covert channel communication

In computer networks Alice and Bob do not have to be the sender and receiver of the overt communication. One or both of them may act as a *middleman* (see Figure 2). If Alice can observe and manipulate an existing overt communication from an innocent sender that reaches Bob, she can insert a covert channel into it. Bob does not need to be the receiver of the overt communication, but merely must be able to observe it to decode the hidden information. If Bob can also alter the overt+covert communication, he can even remove the covert channel preventing further upstream nodes from discovering it. A middleman could be located for example inside a network router or inside an end host's network stack.

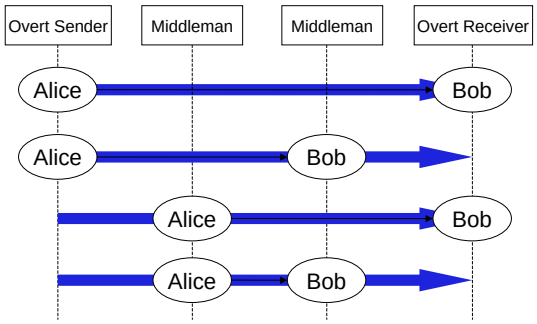

Figure 2. Communication scenarios depending on sender and receiver locations

Traditionally covert channels were classified into storage and timing channels even though there is no fundamental distinction between them [8]:

- *Storage channels* involve the direct/indirect writing of object values by the sender and the direct/indirect reading of the object values by the receiver.
- *Timing channels* involve the sender signaling information by modulating the use of resources (e.g. CPU usage) over time such that the receiver can observe it and decode the information.

B. Countermeasures

Before any action can be taken against a covert channel, it needs to be identified. The identification of a covert channel can be ad-hoc or based on a formal method. A number of formal methods have been developed for identifying covert channels in single host systems [9]. The few proposed formal techniques for identifying covert channels in network protocols are mostly based on an adaptation of one of these techniques [3].

There are two causes of covert channels: design oversights and weaknesses inherent in the system design.

While covert channels caused by oversights may be corrected once discovered, those intrinsic to the system can never be removed without redesigning the system. Therefore, ideally covert channels should be identified and removed during the design phase.

If a covert channel was not removed in the design phase the next best option is to eliminate its possible use. However, this may lead to very inefficient systems, as covert channels can often only be completely removed by replacing automated procedures with manual procedures. Furthermore, covert channels based on the modulation of visible message parameters are inherent in distributed systems, such as computer networks. Therefore, it seems covert channels cannot all be completely eliminated [10].

If a channel cannot be eliminated its capacity should be reduced. What is an acceptable capacity depends on the amount of information leakage that is critical. For example, if the channel is so small that classified information cannot be leaked before it is outdated, then the channel capacity is tolerable. Limiting the channel capacity is often problematic, because it means slowing down system mechanisms or introducing noise, which both limit the performance of the system.

Any covert channels that cannot be removed should be audited. Auditing acts as deterrence to possible users of the channel. Covert channels with capacities too low to be significant, or which cannot be audited should at least be documented (e.g. in the protocol specification), so that everybody is aware of their existence and potential threat.

C. Evaluation Criteria

We propose three main criteria for evaluating covert channels in network protocols. These criteria are similar to those used for evaluating steganographic systems [11]:

- *Capacity* determines the maximum capacity of the covert channel in bits per packet of the overt traffic. The capacity in bits per time unit depends on the amount of overt traffic.
- *Robustness* determines how easily the covert channel can be eliminated and how robust it is against channel noise.
- *Stealth* determines how different the modified traffic is from normal unmodified traffic (also referred to as stealth).

Obviously capacity, robustness and stealth are conflicting goals. It is impossible to optimise them all at the same time. Usually a trade-off must be chosen that is best for a particular situation.

III. Goals and Features

This section describes the overall goals and main features of CCHEF. The overall goal of the software is to hide covert data in overt network traffic for various research purposes:

- Evaluate the capacity, stealth and robustness of network covert channels and compare different covert channels in different scenarios.
- Evaluate and compare mechanisms to eliminate, limit the capacity or detect different network covert channels in different scenarios.
- Create test traffic for existing intrusion detection software or firewalls (composed of some overt traffic with one or more hidden covert channels).

It is not a goal for the software framework to actually become (mis)used for circumventing any security measures. Therefore, no attempts will be made to hide the presence of the software itself. The sender and receiver are normal user space applications and if root privileges are needed for the execution, applications need to be started as root. This allows us to focus on the actual covert channel methods, avoids facilitating possible misuse of the software, and improves the portability of CCHEF (since techniques to hide executables are very operating system dependent).

There are two different types of scenarios we want to support:

1) CCHEF should work in real networks with real overt traffic. This allows evaluating covert channels across real networks and is mandatory for the testing existing intrusion detection software or firewalls.
2) Usually testing with real traffic is restricted to controlled testbeds where it is almost impossible to generate a realistic traffic mix from a larger number of hosts. Therefore, CCHEF should also be capable of running on a single host emulating the use of covert channels with overt traffic from traffic traces.

This following list describes the main features of CCHEF:

- CCHEF allows evaluating covert channels regarding their capacity, robustness and stealth.
- CCHEF supports both storage and timing channels.
- CCHEF supports covert channels in the IP protocol and in higher-layer network protocols (e.g. TCP, HTTP). It is planned as future work to extend CCHEF to also support link-layer channels.

- CCHEF currently only supports IPv4. Support for IPv6 may be added in the future.
- CCHEF is very flexible and extensible. It is possible to create new covert channel encoding modules without having to write or modify any code of the framework itself. Furthermore, it is possible to easily modify or add new code for encryption, authentication, framing and reliably transport of the covert data.
- In general the source code of CCHEF was written to be as portable as possible, so that CCHEF can be used on different operating systems. However, the primary development platform is Linux and currently some functions only work on Linux.
- CCHEF is efficient enough (in terms of CPU and memory usage) to handle real 'realistic' overt traffic (realistic mix of protocols and packet sizes) of up to 100 Mbit/s (Fast Ethernet) and trace files of up to few hundred million packets.

IV. Architecture

This section describes the architecture of CCHEF. First we present an overview of the architecture and the different layers of the covert communication. Then we discuss parts of the architecture in more detail: input/output devices, selection of overt packets, covert channel encoding/decoding modules and error simulation/emulation.

A. CCHEF Modes

Figure 3 shows how CCHEF is used for actually transmitting covert information across a network. For clarity the figure only shows the unidirectional channel of Alice sending to Bob, but channels in CCHEF can be bi-directional and therefore Bob could send to Alice at the same time.

Alice and Bob tap into an overt channel, which is a number of traffic flows between Alice and Bob. Alice and Bob can be the sender and receiver of the overt traffic or act as middlemen and use pre-existing overt traffic of unwitting users. Alice intercepts the overt traffic, encodes the covert data and re-injects the overt traffic with the covert channel embedded into the network. Bobs intercepts the overt traffic, decodes the covert information and possibly removes the covert channel.

Figure 4 shows how CCHEF can be used to evaluate covert channels based on overt traffic from trace files. In this case covert information is not actually send across a network; Alice and Bob are a single entity. For each overt packet read from the trace file CCHEF first embeds the

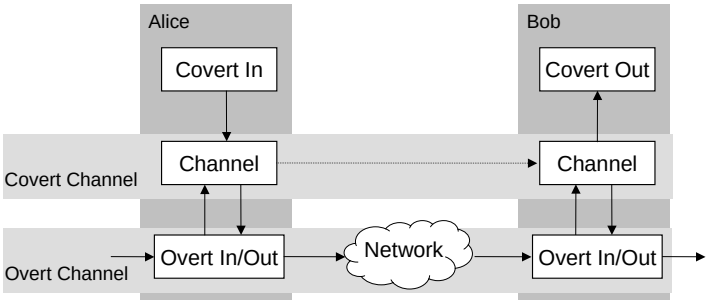

Figure 3. CCHEF used to transmit information across the network (Alice sending to Bob)

covert information, optionally simulates noise and then decodes the covert information from the packet straight away. The received information is compared with the sent information and statistics are computed (e.g. channel error rate). The modified trace can be stored for later use, such as testing firewalls or intrusion detection systems.

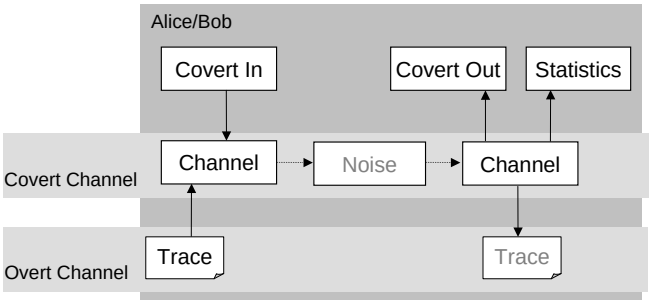

Figure 4. CCHEF used to evaluate covert channels based on overt traffic from trace files

B. Main Modules

Figure 5 shows the modules of CCHEF. The heart of CCHEF is the *Channel* module that interfaces with multiple *Device* modules. Covert data to be send is read through a *Covert In* device while received covert data is passed to a *Covert Out* device. The *Overt In/Out* device intercepts and re-injects IP packets from the network to be used as carrier for the covert information. The *Selection* module selects which packets of the overt traffic are used to embed the covert channel.

The Overt In/Out module intercepts overt packets in the send direction. A subset of packets (the carrier traffic) is chosen by the Selection module and passed on to the Channel module. Packets not selected are re-injected into the network immediately. The Channel module encodes covert data into the carrier traffic and passes the modified packets back to the Overt In/Out device, which then re-injects them back into the network.

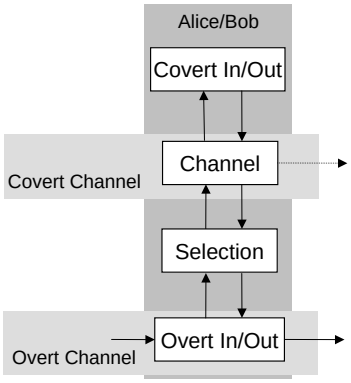

Figure 5. CCHEF main modules

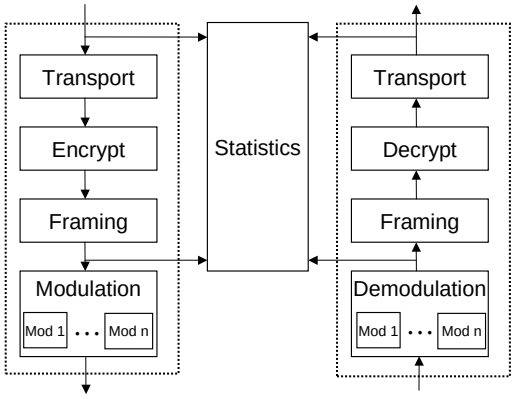

Figure 6. CCHEF channel module functions at sender and receiver

Overt packets arriving in the receive direction are intercepted by the Overt In/Out module and passed on to the Channel module. The Channel module decodes any covert information present in the packets (and if possible removes the covert channel). CCHEF also supports passive receivers that work on copies of the original packets avoiding delay of the overt traffic, if removing the covert channel is not desired or not possible. In this case the Overt In/Out device intercepts not the actual packet, but a copy of the packet.

The Channel module performs all the functions necessary for covert communication such as modulation/demodulation of the covert data, segmenting the covert bits into blocks (framing/de-framing), encryption/decryption and reliability transport (see next sub section). A configuration module configures all devices and the channel itself. Configuration information is mainly taken from a configuration file (see Section V).

C. Channel Module

The Channel module is composed of multiple sub modules, each representing a communication layer of the covert channel. Figure 6 shows the details of the channel layers on the sender (left) and receiver (right). There are four main modules in CCHEF: modulation, framing, encryption and transport. The modulation, framing and transport modules are similar to the physical, link and transport layers of the OSI model. CCHEF does only support a connection between one covert sender and one or multiple covert receiver(s) and therefore the equivalent of the OSI network layer does not exist. Note that Framing, Encryption or Transport modules can also be bypassed if their particular function is not required.

The *Modulation* modules are responsible for modulating/demodulating the covert bits into the overt packet stream and bit synchronisation (if needed). Each module implements a certain covert channel technique and provides modulation (sending) and demodulation (receiving) functions. Multiple modulation modules can be used in parallel to encode covert information using different covert channel techniques simultaneously. However, Alice and Bob must use the same set of modules with the same parameters in the same order. The implemented modules are described in Section IV-F.

The *Framing* module is responsible for framing (including byte synchronisation). Each Framing module provides a framing (sender) and de-framing (receiver) method. Currently we have implemented Start of Frame (SOF) based framing based on the technique used by HDLC and checksum-based framing based on the CRC32 function used by ATM. Overt traffic is always present, but Alice may not have covert data to be send. To improve stealth Alice does not modify the overt traffic when idle i.e. she does not send an idle signal. This must be taken into account by the framing module. For SOF-based framing the current frame ends either when a new SOF is received or when the number of bits received equal the expected frame length (either fixed or signalled by a frame length header field).

The *Encryption* module encrypts/decrypts the covert data. Currently only a simple XOR-based algorithm is implemented. The main reason for having this module is not to actually secure the covert data, as CCHEF has been developed for the sole purpose of research. But encryption can change the distribution of the input data (the distribution of 0 bits and 1 bits), which may affect the characteristics of the channel. Often theoretic models assume uniform random distributed input data. For example, the maximum capacity of the binary symmetric channel is achieved only under this assumption [12].

The *Transport* module is responsible for loss detection, error correction, retransmissions, and flow control (if needed). The transport module also performs segmentation of the input data into packets. Note that the frame size is equal to the packet size, as lower layers do not perform any further segmentation. At the moment there is only one simple Transport module implemented. This module uses fixed-size packets, because then a packet length header field is not needed. If a packet cannot be completely filled, because there is no covert data (within some time limit), it is padded with zeros. The module uses sequence numbers to detect loss of covert packets.

During operation Alice and Bob compute statistics, for example the number of covert bits sent or received. Also Alice and Bob log actual copies of the bit stream for analysis purposes, such as computing the bit error rate. Firstly, Alice logs the bit stream before modulation and Bob logs the bit stream after demodulation (transport layer). Secondly, Alice logs the bit stream before transport and Bob logs the bit stream after transport (payload layer). If Alice and Bob work on overt data from traffic traces the bit streams can be compared during operation. If a network separates Alice and Bob, bit streams are stored and can be compared offline.

D. Device Modules

Devices are used to input and output covert data, and to intercept and re-inject overt traffic into which covert information is encoded. There are a number of different device modules. First we describe the function of the devices and then table I summarises the purposes of the different devices.

The *Random* device generates a uniform random data stream (the probability of 0 bits and 1 bits is equal). This device is used for avoiding a bias that specific input data may introduce to the analysis. For example, the amount of bit stuffing done by the SOF framer depends heavily on the input data. The *Null* device writes data to /dev/null. This device is useful when the received data should be ignored. The *Text* device reads from a text file or outputs to a text file.

The *Tunnel* device reads and writes IP packets from and to a tunnel network device. This enables CCHEF to tunnel IP packets of any networked application across the covert channel. The *Netfilter Queue* device instruments the Netfilter queue framework inside the Linux kernel [13]. This framework consists of a number of hooks in the packet processing routines of the kernel. Packets are intercepted and delivered to a userspace application. The userspace application can modify the packets and re-inject them back into the kernel (or drop them). Netfilter only allows write access to the IP header and data. Therefore currently it is not possible to embed covert channels in link layer protocols. We plan to add another device for this at a later stage. The Tunnel and Netfilter Queue devices work under only under Linux.

The *Libtrace* device reads and writes packets from and to various trace file formats e.g. libpcap format, Endace Record Format (ERF), and some older formats used by DAG measurement cards [14]. Its main purpose is to read overt traffic from trace files and to create trace files with embedded covert channels that could be used to test firewalls or intrusion detection systems. However, it could also be used to send recorded IP packets across the covert channel. It is based on the libtrace library [15].

Table I
DEVICE MODULES AND THEIR FUNCTION

Device	Covert In	Covert Out	Overt In/Out
Random	yes	no	no
Null	no	yes	no
Text	yes	yes	no
Tunnel	yes	yes	possibly
NetfilterQueue	possibly	possibly	yes
Pcap	possibly	possibly	yes
Libtrace	possibly	possibly	yes

E. Packet Selection Modules

The packet selection module determines which overt packets are used as carrier for the covert channel. The module is composed of a number of sub modules: classifier module, flow grouping module, and selector module.

Covert packets are obtained either via the Netfilter Queue device (real network) or via the Libtrace device (traffic trace analysis). The *Classifier* module only passes on packets that match a set of configurable rules. The rules use the standard 5-tuple of source/destination IP address, protocol, and UDP/TCP ports for packet matching. Non-matching packets are re-injected into the network straight away. The use of this module is to limit the covert channel to overt traffic flowing between the specified sender/receiver pair(s).

Then *Flow Grouping* module groups packets into bi-directional flows according to the 5-tuple. Each flow is identified by a 'unique' flow ID. The flow ID is a 64-bit integer so it will wrap around eventually. Packets are grouped into flows because some covert channel techniques can only encode data relative to flows. Flows are ended by a timeout or by a TCP connection teardown,

in which case any flow state is deleted. The first packet of a flow defines the forward direction. Packets with IP addresses and ports reversed are going in backward direction.

Then packets are passed to the *Selector* module. The purpose of this module is to only select specific packets within a flow or across flows to be used as carrier for covert information. For example, the selector module could be used to only encode the covert channel in some packets or flows determined by a secret shared between Alice and Bob. Packets meeting the selection criteria are passed on to the modulation module. Modified packets (including covert data) are re-injected into the network by the Overt In/Out device. Figure 7 depicts the process of selecting overt packets used as carrier for the covert channel.

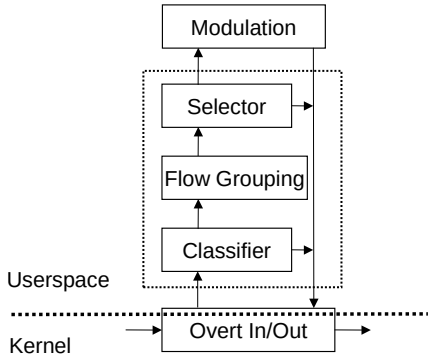

Figure 7. Selection of overt packets to be used as carrier for covert data

F. Modulation Modules

Modulation modules are responsible for embedding covert data into the overt traffic and for decoding (and possibly removing) covert data from the overt traffic. Modulation modules are fairly independent of CCHEF (realised as shared libraries). This means new modules can be added without the need to modify or recompile CCHEF.

A module has global constructor and destructor functions to initialise and destroy global state. A module also has per-flow initialise and destroy functions, which are called each time a new flow starts or an existing flow ends (as determined by CCHEF's Flow Grouping module). In the per-flow initialise function a module can allocate per flow state. Each time a suitable overt packet arrives, CCHEF calls either the encode or decode function of a module. Whether the encode or decode function is called depends on the direction of the overt packet as determined by the Classifier, i.e. if the packet travels from Alice to Bob (forward) or vice versa (backward). CCHEF passes pointers to the packet data, packet meta-data (such as packet arrival timestamps) and the allocated per-flow state (if the module is using per-flow state) to the encode and decode functions.

CCHEF also provides timers for modulation modules. A modulation module can register multiple timers at start-up and every time a new flow starts. Each timer is characterised by a unique ID and a timeout value. When a timer expires CCHEF calls the timeout function of the module. Each time the timeout function is called the module can adjust the timeout value of the expired timer.

Figure 8 summarises the various functions of modulation modules called by CCHEF.

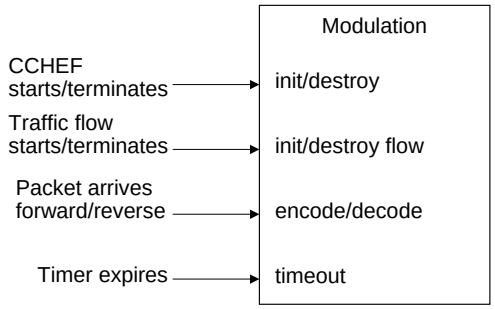

Figure 8. Modulation module functions

We have implemented a number of modulation modules. Several modules have been developed for different techniques to encode covert information in the IP Time to Live (TTL) field [16]. Another module encodes the covert channel in the IP ID fields as described in [17]. Furthermore, there are two modules for encoding covert information into the timing of packets: one module encodes covert bits as absolute delays and the other modules uses the encoding scheme proposed in [18].

G. Error Simulation Modules

CCHEF can simulate channel errors. This is most useful in trace-file mode (see Figure 4). The error simulation is done at the sender-side after the bit modulation. An error module has two functions, which are called with a pointer to the current overt packet and its meta-data. The *pre* function is called prior to modulation. This function allows the error module to learn the original unmodified packet data. The *post* function is called after the modulation. This function enables the error module to modify the overt packet with embedded covert data and thus simulate any kind of error characteristics of the covert channel.

The error simulation of CCHEF was primarily designed to simulate channel errors caused by the modification of data fields in the packet (storage channel) or timing noise (packet timing) on the path between Alice and Bob. However, lost or reordered IP packets also introduce errors in the covert channel (unless the covert channel is based on a reliable transport protocol such as TCP). Currently CCHEF cannot simulate errors related to the loss or reordering of IP packets, but we plan including this in future versions.

V. Configuration

This section illustrates how CCHEF is configured. For further information on how to configure and use CCHEF the interested reader is referred to the user manual [6]. An XML configuration file controls the behaviour of CCHEF. The configuration file is divided into several parts:

- The main section defines general settings e.g. the name of the log file and how detailed the logging is.
- Module specifications define the modulation modules.
- Device specifications define the input/output devices.
- Encryption specifications define encryption/decryption techniques used for the covert data.
- The packet selection specifications define what packets are used to embed covert information.
- Framer specifications define how covert data is split into frames and how frames are decoded at the receiver.
- Transport specifications define how covert data is transported e.g. error detection and recovery.
- Noise specifications define if and how noise is simulated.
- Finally, the covert channel is defined by specifying the devices for the input and output of covert data, the source of overt packets used as cover, and the encryption, framing, transport, packet selection techniques used.

In each part configuration information is specified as preferences (PREFs). Preferences have a name, a value and (optionally) a type specification. The following is an example for a preference:

```
<PREF NAME="Verbose" TYPE="UInt8">4</PREF>
```

Table II shows the existing types. The type is an optional attribute (used for checking the values). If no type is specified, the default type of string is assumed.

Table II
Existing data types for configuration preferences

Type	Explanation
UInt8	8 bit unsigned integer
SInt8	8 bit signed integer
UInt16	16 bit unsigned integer
SInt16	16 bit signed integer
UInt32	32 bit unsigned integer
SInt32	32 bit signed integer
UInt64	64 bit unsigned integer
SInt64	64 bit signed integer
Float	Single precision floating point
Double	Double precision floating point
Bool	Boolean (yes or no)
String	Character string
IPAddr	IPv4 address in dotted decimal notation or domain name

The following shows a configuration file with only the main part and the covert channel definition. The other parts of the configuration are explained later.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE CONFIG SYSTEM "config.dtd">
<CONFIG>
<MAIN>
  <!-- general settings -->
  <PREF NAME="Verbose" TYPE="UInt8">4</PREF>
   <PREF NAME="LogFile">/home/szander/src/
     cchef/install/var/log/cchef.log</PREF>
</MAIN>
<!-- modulation modules -->
<!-- devices -->
<!-- encryption -->
<!-- packet selection -->
<!-- framing -->
<!-- transport -->
<!-- noise simulation -->
<!-- covert channel -->
<CHANNEL NAME="channel1">
  <PREF NAME="CovertIn">InFile</PREF>
  <PREF NAME="CovertOut">OutFile</PREF>
  <PREF NAME="Cover">NFQueue</PREF>
  <PREF NAME="PacketSelection">All</PREF>
  <PREF NAME="Cryptor">XOR</PREF>
  <PREF NAME="Framer">SOF</PREF>
  <PREF NAME="Transport">Simple</PREF>
  <PREF NAME="Noise">TTLRandom</PREF>
  <PREF NAME="Modules">ttl</PREF>
</CHANNEL>

</CONFIG>
```

A channel must specify Cover and either CovertIn or CovertOut (unidirectional channel) or both (bidirectional channel) using device names of devices specified in the device section of the configuration. It must also specify at least one module under Modules using the module name(s) as specified in the module section of the configuration. Multiple modules can be specified

by separating the names with spaces. If multiple modules are specified covert data is encoded by the modules in the order they are specified. This means sender and receiver configuration files must specify modules in the same order; otherwise the communication will not work.

PacketSelection, Cryptor, Framer, Transport and Noise specifications are optional. By default all overt packets will be used to encode covert data, and the SOF framer and simple transport module will be used. By default no encryption will be used and no noise is simulated.

The following specifies a modulation module. Each module must have a unique name. The preferences are module-specific [6].

```
<MODULE NAME="ttl">
  <PREF NAME="BitsPerPacket">1</PREF>
  <PREF NAME="Delta">1</PREF>
</MODULE>
```

A device specification must have a unique name and have the Type preference set to an existing device (see section IV-D for a list of existing devices). Other preferences are device-specific [6].

```
<DEV NAME="InFile">
  <PREF NAME="Type">File</PREF>
  <PREF NAME="Filename">send.txt</PREF>
</DEV>
```

Encryption/decryption specification must have a unique name and Type must be set to an existing module. Currently the only existing module is *XOR* [6].

```
<CRYPT NAME="XOR">
  <PREF NAME="Type">XOR</PREF>
  <PREF NAME="Key">geheim</PREF>
</CRYPT>
```

A framer specification must have a unique name and the Type preference must be set to an existing module. Currently two framing methods are implemented. The *SOF framer* uses a start of frame byte and bit stuffing whereas the *CRC framer* identifies frames based on a CRC32 checksum [6].

```
<FRAMER NAME="SOF">
  <PREF NAME="Type">SOF</PREF>
</FRAMER>
```

A transport specification must have a unique name and the Type preference must be set to an existing module. Currently the only transport module implemented is the *Simple* module [6].

```
<TRANSPORT NAME="Simple">
  <PREF NAME="Type">Simple</PREF>
  <PREF NAME="BlockSize">16</PREF>
  <PREF NAME="Parity">0</PREF>
</TRANSPORT>
```

The noise specification is optional and is only present for testing covert channels with overt data from trace files. A noise specification must have a unique name and the type must be set to an existing noise module. Currently two noise modules exist for Time to Live (TTL) covert channels. The *TTL* module emulates noise based on TTLs from trace files and the *TTLRandom* module generates artificial noise based on a Gaussian distribution [6].

```
<NOISE NAME="TTLRandom">
  <PREF NAME="Type">TTLRandom</PREF>
  <PREF NAME="StdDev">0.5</PREF>
</NOISE>
```

The packet selector specification is optional. By default all available packets are used to embed covert data. A packet selection specification must have a unique name and the Type preference set to an existing module. Currently the *All* module selects all packets and the *Hash* module samples packets with specified probability [6].

```
<SELECTOR NAME="All">
  <PREF NAME="Type">All</PREF>
</SELECTOR>
```

VI. Conclusions and Future Work

In this paper we describe the Covert Channels Evaluation Framework (CCHEF). CCHEF can be used to evaluate the capacity, stealth and robustness of covert channels in network protocols. Since it creates covert channels it can also be used for generating the data needed to evaluate countermeasures against network covert channels or for testing existing firewalls or intrusion detection software.

CCHEF can be used in real networks with real overt traffic, but can also emulate covert channels using overt traffic from trace files. CCHEF supports both covert channels hidden in data fields (storage channels) and the timing of packets (timing channels). The architecture of CCHEF is very flexible and extensible. New covert channels modules can be added without the need to modify the framework itself. Furthermore, new mechanisms for covert channel security (authentication, encryption), framing and (reliable) transport can easily be added.

CCHEF is still under development and a number of desired features remain future work. For example, the simulation of IP packet loss and reordering and link layer covert channels have not been implemented yet. Furthermore, we plan to fully optimise CCHEF and measure its performance under a variety of conditions.

REFERENCES

[1] B. Lampson, "A Note on the Confinement Problem," *Communication of the ACM*, vol. 16, pp. 613–615, October 1973.

[2] S. Zander, G. Armitage, P. Branch, "Covert Channels and Countermeasures in Computer Network Protocols," *IEEE Communications Magazine*, vol. 45, pp. 136–142, December 2007.

[3] S. Zander, G. Armitage, P. Branch, "A Survey of Covert Channels and Countermeasures in Computer Network Protocols," *IEEE Communications Surveys and Tutorials*, vol. 9, pp. 44–57, October 2007.

[4] G. Fisk, M. Fisk, C. Papadopoulos, J. Neil, "Eliminating Steganography in Internet Traffic with Active Wardens," in *Proceedings of 5th International Workshop on Information Hiding*, October 2002.

[5] M. Van Horenbeeck, "Deception on the Network: Thinking Differently About Covert Channels," in *Proceedings of 7th Australian Information Warfare and Security Conference*, December 2006.

[6] S. Zander, "CCHEF – Covert Channels Evaluation Framework User Manual," May 2008. http://caia.swin.edu.au/cv/szander/cc/cchef/cchef-user-manual.pdf.

[7] G. J. Simmons, "The Prisoners' Problem and the Subliminal Channel," in *Proceedings of Advances in Cryptology (CRYPTO)*, pp. 51–67, 1983.

[8] US Department of Defense Standard, "Trusted Computer System Evaluation Criteria," Tech. Rep. DOD 5200.28-STD, US Department of Defense, December 1985. http://csrc.nist.gov/publications/history/dod85.pdf.

[9] V. Gligor, "A Guide to Understanding Covert Channel Analysis of Trusted Systems," Tech. Rep. NCSC-TG-030, National Computer Security Center, November 1993. http://www.radium.ncsc.mil/tpep/library/rainbow/NCSC-TG-030.html.

[10] I. S. Moskowitz, M. H. Kang, "Covert Channels - Here to Stay?," in *Proceedings of 9th Annual Conference on Computer Assurance*, pp. 235–244, 1994.

[11] N. Lucena, J. Pease, P. Yadollahpour, S. J. Chapin, "Syntax and Semantics-Preserving Application-Layer Protocol Steganography," in *Proceedings of 6th Information Hiding Workshop*, May 2004.

[12] T. M. Cover, J. A. Thomas, *Elements of Information Theory*. Wiley Series in Telecommunications, John Wiley & Sons, 1991.

[13] H. Welte, "Netfilter Queue Library." http://www.netfilter.org/projects/libnetfilter_queue/.

[14] Endace Limited, "DAG Network Monitoring Cards." http://www.endace.com/what-we-do/dag-network-monitoring-cards/.

[15] WAND Network Research Group, "libtrace – A Library for Trace Processing." http://research.wand.net.nz/software/libtrace.php.

[16] S. Zander, G. Armitage, P. Branch, "An Empirical Evaluation of IP Time To Live Covert Channels," in *Proceedings of IEEE International Conference on Networks (ICON)*, November 2007.

[17] C. H. Rowland, "Covert Channels in the TCP/IP Protocol Suite," *First Monday, Peer Reviewed Journal on the Internet*, July 1997.

[18] G. Shah, A. Molina, M. Blaze, "Keyboards and Covert Channels," in *Proceedings of USENIX Security Symposium*, August 2006.