Covert channels analysis in TCP/IP networks

Pierre Allix

kototamo at gmail dot com

IFIPS School of Engineering

University of Paris-Sud XI, Orsay, France

June 2007

"Ce qui est essentiel est invisible pour les yeux"

Antoine de Saint-Exupéry, Le Petit Prince.

Table of Contents

Résumé

Les canaux cachés utilisent des communications furtives afin de compromettre les politiques de sécurité des systèmes. Ils constituent une menace importante pour la sécurité puisqu'ils peuvent être utilisés afin d'exfiltrer des données confidentielles des réseaux. Les protocoles TCP/IP sont utilisés tous les jours et sont sujets aux problèmes des canaux cachés. Le but de cet article est de donner un aperçu des canaux cachés dans les réseaux TCP/IP. Nous décrivons brièvement les protocoles TCP et IP, présentons les différents types de canaux cachés et les méthodes pour les mettre en place dans les réseaux TCP/IP; nous étudions ensuite les méthodes existantes pour détecter et éviter les canaux cachés. Les méthodes statistiques, les réseaux neuronaux, les machines à vecteurs de support et les normaliseurs de traffic sont présentés. Quelques nouvelles idées sur les techniques de tri de paquets, de comptage de paquets et d'évasion des normaliseurs de traffic sont données.

Abstract

Covert channels use stealth communications to compromise the security policies of systems. They constitute an important security threat since they can be used to exfiltrate confidential data from networks. TCP/IP protocols are used everyday and are subject to covert channels problems. The aim of this paper is to give an overview of covert channels in TCP/IP networks. We briefly describe the TCP and IP protocols, present the different types of covert channels and the methods to set them up in TCP/IP networks; then we study the existing methods to detect and avoid covert channels. Statistical methods, neural networks, support vectors machines and traffic normalizer are presented. Some new insights on packet sorting techniques, counting covert channels and traffic normalizers evasion are given.

Keywords

Covert channels, TCP, IP, computer security, networking, detection, protection, analysis, neural networks, support vector machines, traffic normalizers, counting covert channels, packet sorting.

Acknowledgements

The author is really grateful to the German team of Prismtech. Thanks particularly to Sebastian Staamann, Marcus Wittig and Christoph Becker.

Thanks to Nicolas Favre-Félix for the first rereading and corrections.

Intellectual Property

Introduction

Networks based on the TCP and IP protocols are, for most of us, part of our everyday life. We use them for professional and personal needs: to communicate with customers or with friends, to buy presents, to send money, to retrieve information or cultural goods etc. The usages are endless and while the security of our computers has increased these last years, the security of our networks has only slightly been enhanced. Lots of security problems are still present, especially in enterprise and private networks. Covert channels are not the most well-known source of risks, and are in fact totally ignored by the public, but they constitute a real threat.

Several definitions of covert channels exist: a covert channel is “a mechanism that can be used to transfer information from one user of a system to another using means not intended for this purpose by the system developers” [Hu95]. [NCSC TCSEC] defines a covert channel as “any communication channel that can be exploited by a process to transfer information in a manner that violates the system's security policy”. Covert channels implementations try to be, or are, stealth.

Covert channels pose a problem for highly secure environments such as government agencies and military ones. In multilevel security environments where users with high security levels must not be able to pass information to users with lower security levels, covert channels can be used to circumvent such policies. In a more classical environment, covert channels can be used by an attacker to communicate stealthy with a compromised machine, thus complicating the detection of the attack.

In this paper we give a brief overview of the TCP and IP protocols, then present the different types of covert channels and methods to set them up in the TCP and IP protocols, and study the existing methods to detect and avoid covert channels. We finish by seeing how to avoid and manage the covert channels threat.

1 Covert channels in TCP/IP

1.1 Presentation of TCP/IP

TCP and IP are old protocols since they were invented in the 70's and were approved by the Department of Defense (DOD) for the ARPANET in 1982. Now the Internet and most of current networks still rely on TCP and IP to operate. They are efficient but lack security features.

TCP/IP are always referred to together because they are in most cases used together, but they are two different protocols: IP, the Internet Protocol and TCP, the Transmission Control Protocol. Like most networks, TCP/IP networks are built on a layered architecture. There are four layers for TCP/IP networks. From the top to the bottom, we have:

- Application layer: this layer is used by end user services. For instance HTTP, the protocol used to access the content of the Web, or POP, the protocol used to retrieve emails, are encoded in this layer.
- Transport layer: this layer handles the flow of information between systems. It manages connections from a system to another.
- Network layer: this layer transmits packets between systems.
- Link layer: this layer provides the communication with the network interface. For instance, on most local networks, the protocol used to send data on the network is Ethernet.

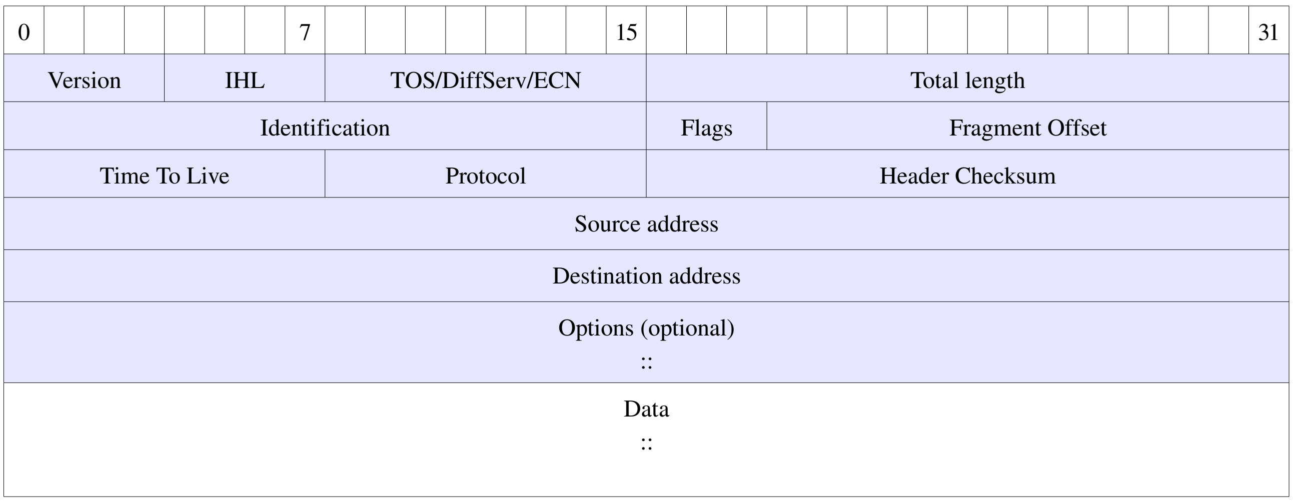
These layers constitute a “stack” and implementations of TCP/IP are often called TCP/IP stacks.

TCP is defined in [RFC793] and IPv4, the current version of IP, in [RFC791]. A good introduction to TCP/IP, a bit outdated, is available in [Ste93]. We do not explain the whole protocols in this section but only the necessary information to understand the rest of this paper.

1.1.1 Internet Protocol

IP, the Internet Protocol, uses the network layer. It is an unreliable, connectionless protocol. This mean that there is no guaranty that an IP packet will not be lost. The protocol does not maintain any state or information between the successively emitted packets. How can we have reliable communications if IP is unreliable ? TCP is the answer: it uses IP and is designed to be reliable. The only role of IP is to transmit packets from a machine to another. A packet is just a piece of data, associated with metadata. These metadata indicate where the data should be transmitted, from where they come etc. The metadata section of a packet is called a header.

The IP header is represented coloured in the following figure:



The '::' symbol indicates that the field can be larger than 32 bits. The IP header, without options, has a length of 20 bytes.

The *version* field indicates the version of the protocol used. The current version of IP is the version 4, so this field will have the value 4.

The *IHL (Internet Header Length)* field indicates the number of 32 bit words in the header, with the options.

The *Type Of Service (TOS)* field is now used for Quality of Service (QoS with DiffServ) to distinct priority data from the other, for instance for Voice over IP. It can also be used for congestion management (ECN extension).

The *Total length* field indicates the total size of the packet in bytes.

The *Identification* field is used for fragmentation. MTU (Maximum Transfer Unit) is the maximum size for a packet that a device accepts. When a packet is bigger than the MTU it has to be fragmented into several smaller packets. Each one contains an identification field that is used by the receiver to reassemble the original packet, each fragmented packet of the same original packet having the same identification number.

The *Flags* field contains three flags, each coded with a bit:

- The first one is reserved and must be set to zero.

- The second one, DF (Do not fragment), indicates that a packet should not be fragmented. A packet that has to be fragmented and that has the DF flag set is discarded.
- The third one, MF (More Fragments) is set on each fragment of a fragmented packet but the last.

The *Fragment Offset* offset field indicates the offset of a fragment in the original packet, in 8-byte block unit.

Fragmentation should rarely occur since MTU discovery techniques now exist.

The *Time To Live (TTL)* is decremented by one each time a packet is forwarded by a hop. When the TTL is zero, the packet is discarded. This prevents packets from persisting in the network, for instance if there are some routing problems.

The *Protocol* field indicates which kind of protocol is used in the data of the packet. 1 is for ICMP, 6 for TCP, 7 for UDP...

The *Checksum* field is the checksum of the header. It is used to detect errors during the transmission of the packet. This checksum is verified at each hop and recomputed because the TTL field change at each hop too. The data are not part of the checksum, and protocols of the transport layer may have their own checksum field. The algorithm for the checksum is simple:

“The checksum field is the 16 bit one's complement of the one's complement sum of all 16 bit words in the header. For purposes of computing the checksum, the value of the checksum field is zero.” [RFC791]

The *Source address* field indicates the source address of the machine that emits the packet.

The *Destination address* field indicates the address of destination.

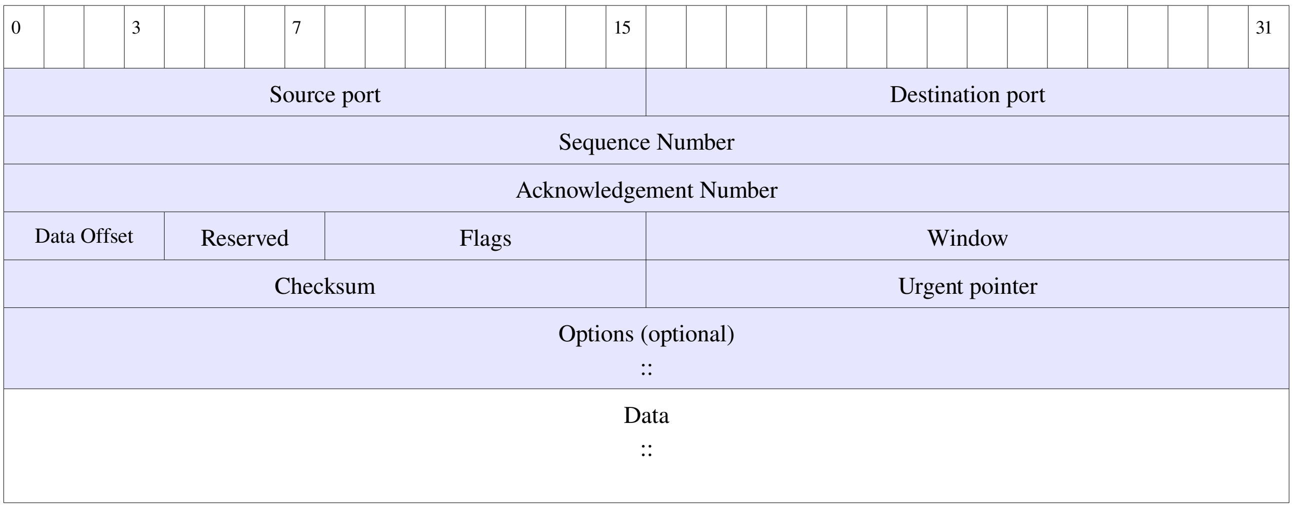
The *Options* is an additional optional field. It can contain information about security, routing etc. It is rarely used. The options are always aligned to a 32 bits boundary, padding is used to maintain this property.

The *Data* field contains data of the layers above.

1.1.2 Transmission Control Protocol

TCP handles the communication between two machines. It is a reliable protocol. Data loss or corruption are detected and the packet is emitted again if necessary.

The following figure describes the TCP header:



Source port and *destination port* are used to identify a connection: the quadruplet (source port, source address, destination port, destination address) is unique for each connection.

Each protocol has its own port: for instance HTTP servers use the port 80, FTP server the port 21 etc. When a client makes a connection to a HTTP server, it sends data to the port 80 of the server machine and receives data on some port of its machine.

The *Sequence Number* (SEQ#) field is used to identify, in the flow of information, which bytes are being sent. The stream during a connection is seen as a continuous stream of bytes. SEQ# points to each part of the stream being send.

For instance if 31337 bytes of data have to be sent during a connection, several TCP segments may be sent on the network. Once the connection is established the first segment of data will have a sequence number of 1. If 457 bytes of data have been sent, the second segment will have a SEQ# of 458, etc.

In practice the first sequence number, called the **Initial Sequence Number** (**ISN**), is randomly chosen for each connection for each machine. So in our example the first segment (not in the connection establishment) will have a sequence number of x + 1, where x is randomly chosen, the second one x + 458 bytes and so on. There is an offset but since we know the ISN we can always know which byte is pointed by the SEQ#.

The *Acknowledgement Number* (ACK#) field indicates which bytes of the stream have been received. In the previous example, once a machine has sent 457 bytes of data, the second machine will send a TCP segment with ACK value of 458 (we assume its ISN is 0), notifying thus that the 458th and following bytes are awaited.

To hijack a communication between a server and a client, an attacker must guess the SEQ#s of the server to send packets with valid ACK numbers. If the attacker sends packets with the source address of the client and valid ACK#, the server will “think” that the packets really come from the client.

A few years ago, ISN were not randomly, or not randomly enough, chosen. Attackers could guess the value of the ISN and inject packets in the connection between the client and the server.

The *Data Offset* field indicates the number of 32 bit words in the header, options included.

The *reserved* field is reserved for the future and is not used.

The *flags* field is composed of eight bits:

- CWR and ECE are used for congestion management.
- URG is set to indicate that the *Urgent Pointer* field is valid.
- ACK is set when the ACK# is used.
- PSH is set to indicate that the received data should be passed immediately to the application layer and not be buffered.
- RST is set to reset the communication.
- SYN is set to initialize a communication and synchronize SYN# and ACK# between a client and a server. This will be detailed in the next section.
- FIN is set to indicate that the sender has no more data to send.

The *Window* field indicates the number of bytes, starting with the byte specified in the ACK#, that the sender is willing to accept.

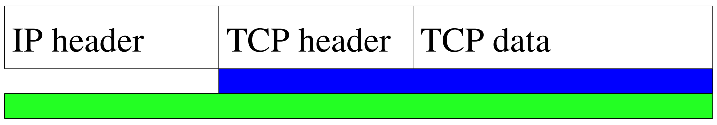
The *Checksum* field is used for error-checking and is applied to the header and the data. The same algorithm as IPv4 is used.

The *Urgent Pointer* field is an offset: added to the beginning of the data it points to the last byte of the urgent data. It is significant only when *URG* is set. Urgent data are defined by the application layer and must be processed quickly. It is rarely used but can be used to indicate “interrupt” type commands, for example with the old, insecure, Telnet protocol.

The *Options* field is optional and can be used for TCP extensions. The options are always aligned to a 32 bits boundary, padding is used to maintain this property.

A common option is the Maximum Segment Size (MSS) that specify the maximum size that the sender wants to receive for segments. Two other options are widespread: the timestamp value (TSV) and the timestamp echo reply value (TSER). TSV contains the value of the time clock of the machine sending the segment, TSER contains the most recent TSV value received. Both these options are used for congestion management.

TCP data are transmitted over IP. The encapsulation schema is like this:



What is called a TCP segment is in blue; the IP packet in green. IP is also encapsulated in another protocol, for instance in a LAN its is commonly over Ethernet.

1.1.3 TCP handshake

Understanding how the TCP connection is established is important to understand the mechanism of covert channels.

To establish a connection, the client emits a first segment with the SYN flag set and with an ISN randomly chosen, in our example 42. If the server accepts the connection, it responds by sending a segment with the SYN and ACK flags set, and an ACK# of ISN + 1. This follows the acknowledgement mechanism described in the previous section. The segment contains also the server SEQ# which is randomly chosen, for instance 31337. The client acknowledges the reception of the segment of the server with a third segment that contains a SEQ# incremented by one, and an ACK# equal to the server's SEQ# plus one.

This process is illustrated with the following figure. It illustrates the establishment of a connection between a machine on a local network with IP address 192.168.1.33 and an Internet HTTP server (www.free.fr). The port of the machine used for the connection is 48666, the port for the server is 80 (that is the port by default for HTTP).

```
|Time     | 192.168.1.33      | www.free.fr       |

First segment:
|0.044    |         48666 > 80 [SYN] S            |TCP: 48666 > 80 [SYN] Seq=42
|         |                                       |
|         |(48666)  ------------------->  (80)     |

Second segment:
|0.082    |         80 > 48666 [SYN, A            |TCP: 80 > 48666 [SYN, ACK] Seq=31337 Ack=43
|         |                                       |
|         |(48666)  <-------------------  (80)     |

Third segment:
|0.082    |         48666 > 80 [ACK] S            |TCP: 48666 > 80 [ACK] Seq=43 Ack=31338
|         |                                       |
|         |(48666)  ------------------->  (80)     |

The connection is established, request of the client to get a Web page:
|0.082    |         GET / HTTP/1.1                |HTTP: GET / HTTP/1.1
|         |(48666)  ------------------->  (80)     |
```

The last line is not part of the connection establishment but the first segment containing the data of the application layer. These data (the string “GET / HTTP/1.1”) ask the server to send the index page of the web site.

1.2 Covert channels

Covert channels for the TCP/IP protocols use the IP or TCP header to convey information in a way that does not disturb existing communication. Covert channels are dangerous because they can be easily implemented, are largely ignored, and can be hard to detect. Moreover, most network administrators or persons in charge of the network security do not try to detect potential covert channels.

Covert channels are sometimes classified into storage covert channels, that use existing data to encode information, and timing covert channels that use the delay of emission between packets to encode data, or any modulation of the available resources.

Covert channels can be active or passive. A passive covert channel use existing communications to transfer information while an active one create packets or communications.

Steganographic techniques that hide data in the payload of the packet are not considered in this paper. They depend on the application protocol used and are not particular to TCP/IP.

1.2.1 Scenarios

We consider two scenarios in this document. In both scenarios communications occur between a client machine A and a server machine B.

In the first scenario an attacker has the control of the machines A and B and sets up a covert channel between them.

In the second scenario an attacker has the control of the machine A, and of another machine C. The machine A communicates with a non compromised machine B and the attacker uses a covert channel between A and B, to indirectly transfer data to the machine C. The machine C is able to examine the network traffic between A and B.

> To examine the traffic, the machine C has to be on the path between A and B, or on a wireless network, or a LAN with a hub, or a LAN hijacked with an ARP spoofing attack... The electronic resonance of devices can even be used with a TEMPEST-like system to get the content of the communications between A and B!

A third scenario is introduced later. These scenarios are just used to illustrate the use of covert channels, other scenarios can be considered.

1.2.2 Covert channels in IPv4

1.2.2.1 Storage covert channels

TCP/IP implementations follow a robustness principle: “be conservative in what you do, be liberal in what you accept from others“ [RFC793]. Invalid packets regarding to the norm can be accepted when they can be meaningfully interpreted. This lead to redundancies: several different packets can be interpreted the same way. This can be used to code information.

For instance an attacker knows that his packets will not be fragmented if theirs sizes are inferior to the MTU. The DF (Do not Fragment) flag can be set or not, this will not affect the communication. The attacker can set a up a covert channel with this protocol: DF set codes a one, DF not set a zero. This is applied in [Ash02]. Other flags can also be used, according to what the server and intermediate hops accept.

A lot of fields in the IP header can be used to convey information. The most obvious technique, described in [Row96], is to hide information in the *identification* field. This field is rarely used and should be set to zero when not used. However TCP/IP stacks follow the robustness principle and non fragmented packets with non zero *identification* fields can be accepted. This allows an attacker to encode 16 bits of data with each packet. Since a lot of packets are usually sent over a network, the amount of data transmitted with such a covert channel can be huge. This technique can be used in conjunction with the previous one. *Identification* field can also encode data when fragmentation is done.

Fragment offset field can also be used when there is no fragmentation. In a more general way, if no fragmentation is needed during the connection, the attacker can voluntarily fragment a packet to code a one, and doing no fragmentation to code a zero.

Some covert channels are more stealth than others. For instance a channel in the fragment offset field can be easily detected since this field is usually set to zero when there is no fragments. The furtiveness of covert channels and methods to detect them will be study in section three.

The *TTL* field can be used: a high TTL value will encode a one, a low value TTL a zero. Since TTL can encode up to 255 values and that most path between machines in the Internet are largely inferior to 40, an attacker can, for instance, send packets with an initial TTL high value of 255 (encodes one), and an initial low value of 142 (encodes 0). If there are 20 hops between A and B, B will receive packets with a TTL of 235 or 122, encoding ones or zeros. [ZaArBr06] used such a technique. A v value can also be chosen, the high TTL value will be v+1 and the low TTL value will be v-1. This is the same schema but less detectable and less reliable.

A more original technique is to use the *Header Checksum* field. The technique is described in [Aba01] and [Aba05]. It uses collisions in the hash algorithm of IP and TCP to create a valid hash that contains information for any packet.

The *Source address* and *destination address* fields are also usable. A packet with a fake source address can encode data, as well as a packet with a fake source destination. In the second case the attacker must sniff the network to retrieve the content of the packet.

A sniffer monitors network data. To sniff means to run a sniffer software to get what is send and received on a network. Sniffers can also be hardware devices.

Data can also be stored into the *options* field of the header, with non-existent options. Valid options can also be used but with non-zero padding that encodes data.

The *data* field itself can store covert channel data when the RST flag is set.

Data can also be added at the end of the *data* field. In scenario two the additional data is unexpected by the server and must be removed before reaching it.

1.2.2.2 Timing covert channels

Timing covert channels use a smart way to encode information: instead of using the different field of the IP header, they use the emission time between packets. IP timing channels are described in [CaBrSh04].

A time interval can be defined: if a packet is sent during the interval, this codes a one, if no packet is sent this codes a zero.

Timing covert channels can also be simply implemented in IP with a binary code. An attacker can module the delay of emission between packets: a short delay between two packets codes a one, a long delay between two packets codes a zero. Non binary codes can also be implemented.

This kind of channel is not reliable since during the communication a hop can delay a packet. Error-correcting codes or CRC in the covert channel can be used to add reliability.

Another method is possible. If an attacker has the control of two machines A and B, each one having a connection to the same server C, the order of the data sent by each can be used as a covert channel. For instance if the machine A sends a packet and then the machine B sends two, this can be interpreted like a 0. If the machine A send two packets and then the machine B one, this can be interpreted like a 1. Complex codes can be established.

Counting covert channels are introduced, outside the scope of TCP/IP, in [Gra94]. Counting covert channels use the number of events to convey information. No papers on counting channels and TCP/IP were found in the literature.

The most obvious counting covert channel that we can imagine sets up a binary code. An arbitrary number λ is chosen. During a connection if the number of transferred data is inferior to λ, this codes a zero, if the number of transferred data is superior or equal to λ, this codes a one.

1.2.3 Covert channels in IPsec

Let us introduce a third scenario: machine A and machine B are not compromised, machine C and D are controlled by an attacker and have access to the traffic between A and B and can modify or inject new packets in the network. The communications between A and B is modified to set up a covert channel, allowing C and D to communicate.

What happens if we add cryptography to the communications between A and B ? If in the first and second scenario, the attacker has the control of the machine A and so we must assume that he can hijack the cryptographic process and modify the IP headers before they are encrypted. In the third scenario this is still possible but more complex: the covert channels must be removed before the packet reaches its destination, otherwise the digital signature verification will fail. Storage covert channels would not be possible. Timing covert channels would always be possible: the attacker can capture, delay and transfer packets to modulate the time between packets. Adding cryptography to the communications reduces the risk of covert channels but does not remove it completely.

IPsec adds authentication and encryption to IP packets and can be used with IPv4. IPsec adds a new potential covert channel described in [Ash02]. In this thesis, the order of IP packets are used to convey information. With n packets we have n! different ways for ordering them and sending them on the network. This can be used to establish a code, each different order for a set of packets encodes a value. We give an example with three packets d1, d2 and d3:

Order			Corresponding coded value
d1	d2	d3	1
d1	d3	d2	2
d2	d1	d3	3
d2	d3	d1	4
d3	d1	d2	5
d3	d2	d1	6

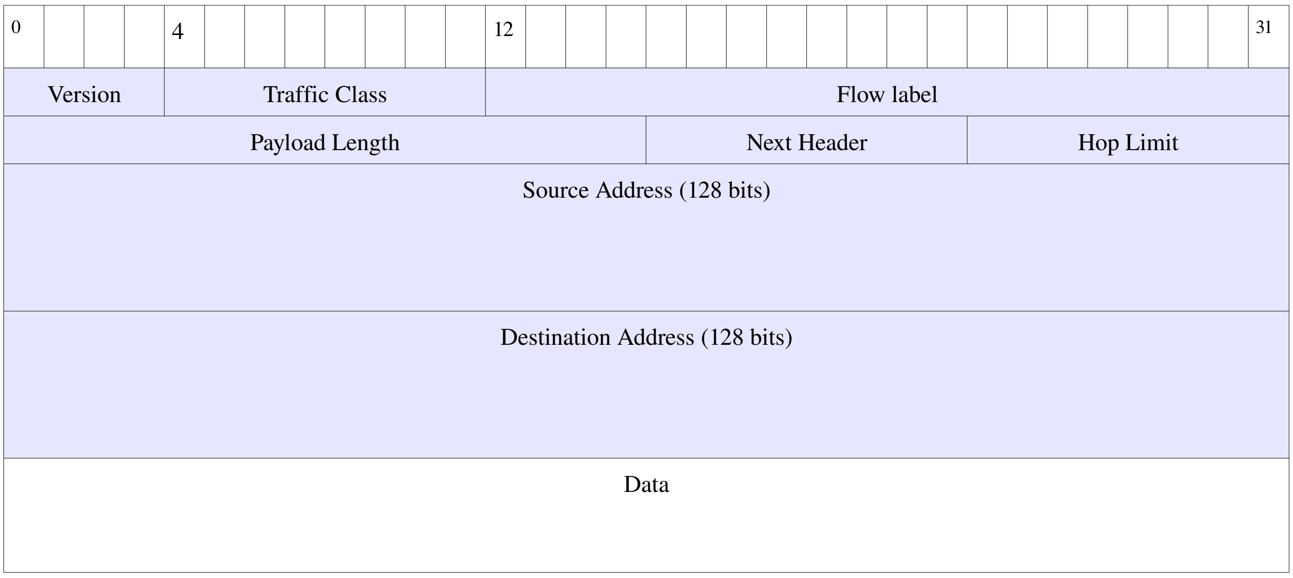
In this example, the attacker receives three packets dx, dy, dz. To discover which value is coded, the original order of the three packets must be known. The Ipsec header has a field that can be used to identify the original order of packets: the sequence number field which is incremented by one with each packet.

The author of [Ash02] writes that the technique is not applicable to TCP. Indeed it is: since TCP sequence number are also strictly increasing (modulo 32 bits), the receiver knows the original order of packets. Packet sorting can be applied to IPv4, based on the content of the sequence number of the TCP header contained in each IP packet, to establish a covert channel.

1.2.4 Covert channels in IPv6

IPv6 [RFC2460] is the successor of IPv4. The protocol is not really spread today, with some exceptions like Japan, but the transition is slowly in progress. IPv4 has to be replaced because the number of IPv4 addresses being attributed is pushing the limits of the total number of IPv4 address available. An IPv4 address is coded with 32 bits, while an IPv6 address is coded with 128 bits, thus allowing more addresses to be attributed. Moreover IPv6 adds new features, like auto-configuration of addresses, and improves the security.

This figure describes the structure of the IPv6 header:



The *Version* field indicates the protocol version, i.e 6.

The *Traffic Class* field can be used to identify different classes or priorities. It is the equivalent to the *Type Of Service* field of IPv4.

The *Flow label* field can be used to QoS (Quality of Service) management. The field is ignored by devices that do not support it.

Payload Length indicates the length of the packet after the header. Extensions headers are part of the header. Extensions headers are additional headers added after the IPv6 header.

The *Next Header* field is equivalent to the Protocol field of IPv4: it indicates the type of header following the IPv6 header.

The *Hop Limit* field is equivalent to the *TTL* field of the IPv4 protocol.

Source Address and *Destination Address* field indicate respectively the source address and the destination address for the packet.

1.2.4.1 Storage covert channels

A real good work has been done in [LuLeCh06] to identify covert channels in IPv6.

The *Traffic Class* field can be used to store data but perturbations can occur if the attacker does not control all the communication path because the value is used by nodes to process the packet.

A fake *flow label* can be created to encode data.

Extra data can be appended at the end of the packet if the *payload length* is modified. This technique is not really a covert channel, as covert channels data use the protocol's expected place to store data.

Since nodes do not process extensions headers, an extension header can be created to store data. This requires modifying the *next header* and *payload length* field.

The *Hop Limit* field can be used as the *TTL* field of the IPv4 for covert channels.

Source Address and *Destination Address* can be used as in the IPv4 protocol.

All theses covert channels and more are detailed in [LuLeCh06].

1.2.4.2 Other covert channels

Timing covert channels are as relevant as for IPv4 and the same techniques can be applied. Counting covert channels can be used. Packet sorting can also be used since IPsec is mandatory for IPv6.

1.2.5 Covert channel in TCP

1.2.5.1 Storage covert channels

We have seen how a TCP connection is established: the SYN flag is set on the first TCP segment sent by the client. We also have seen the RST (Reset) flag that indicates that a connection should be reset. It is obvious that a connection will not be reset when being established, and the RST flag is never set on this first segment. Following the robustness principle or being lazy in their implementation, several operating systems will accept a connection demand with a SYN flag set and a RST flag set . Ambiguities in TCP/IP are described in [Star02]. A client and a server could use this to set up a covert channel: a connection demand with a SYN flag would encode a zero, and a connection demand with a SYN and RST flag would encode a one. Other flags can also be used, even outside the establishment of a connection, depending on what forwarding devices and the server accept.

On most systems, the *source port* field of a TCP connection is chosen between 1024 and 65535. An attacker can fix the source ports of its connections to different values to encode data.

The *destination port* can also be used. If there is no server listening on the port the connection will just be reset but 32 bits would have been transmitted.

The *Initial Sequence Number* (ISN) can also be used. This has been successfully applied with the NUSHU covert channel [Rut04]. NUSHU is a passive covert channel proof of concept for Linux, that modify the ISN of existing segments to code information. Information is coded in ISNs and encrypted with DES, thus ISNs appear to be random.

The *reserved* field can be used.

The *checksum* field can be used with the same method as for IPv4.

The *urgent pointer* is useless when the URG flag is not sent and can thus store 16 bits of data.

The *options* field can be used to store data by creating non existent options. We have seen that the timestamp option is common. The Time Stamp Value (TSV) encodes the timetamp clock value of the sender. In most case the least significant bits of this field appears random, and some covert channels [GiGrLiTi02] used these least bits to encode information.

1.2.6 Covert channels in practice

Someone who wants to set up a storage covert channel must craft his own packet headers. This requires to have administrative privileges. Packets can be crafted with raw sockets or directly inside the kernel. Timing covert channels must be implemented in the kernel. Most storage covert channels can be created with few lines of code by an average programmer.

We have seen that a lot of different covert channels exist in TCP/IP. The data usually used for the protocol itself are used to convey information. Data filtering and data control are thus more complex to implement. Covert channels can be used to violate a security policy and in an easy way. Even the most basic covert channel can be dangerous: if it transmits only a few bytes it will be hard to detect. This dangerous aspect is stressed by the fact that only few networks use covert channels detection mechanisms or mechanisms to avoid covert channels. In the next sections we study these mechanisms.

2 Covert channels detection

2.1 *Simple analysis*

Several covert channels can be detected using a simple analysis of network packets. Let us take the example of covert channels that use the *Urgent Pointer* field when the URG flag field is not used. TCP stacks implementations set this field to zero when URG is not set. This kind of covert channels can be easily detected. The same apply to the MF flags and *Fragment Offset* field of IPv4 when fragmentation is not used.

Identification field (IPv4) generation is implemented with different algorithms [MuLe05]. Some systems have a globally increasing counter, others an increasing per-host counter etc. A covert channel that does not follow these generation schemas will be easily detectable. Nonetheless this requires to know the systems in the studied network to know what *identification* field generation schemas are in use and to compare the *identification* fields in the network with the expected values.

Techniques exist to identify the OS type of a remote system. The TCP/IP stack of each OS is different and each has certain characteristics or anomalies. We can examine the packets sent by a system, study their characteristics and guess the OS of the system. Tools exist to do this, passively or not.

Non-existent options in the *options* field of TCP or IP are also suspicious, as well as a host sending packet with different TTL values or non-zero padding. Sometimes not all bits of ISNs are random: the 15th bit of ISNs generated by OpenBSD is null. A covert channel that ignores these characteristics of TCP/IP stacks can be easily detected.

The reader will find others examples and a captivating discussion about TCP/IP stacks characteristics in [MuLe05].

2.2 *Statistical and likelihood analysis*

2.2.1 Storage covert channels

We have so far covered the subject of random fields. In fact most fields that should be random are not totally random, or are only random on a subset of the space they could occupy.

When using fields that should be random, like ISNs, covert channels try to preserve the randomness to avoid detection. As an example the NUSHU covert channel use DES to have a certain randomness but generated ISNs do not cover the same space as the one generated by Linux and can thus be detected.

For the timestamps technique a random test can be done: with a low bandwidth connection, too much randomness is suspicious.

2.2.2 Timing covert channels

The paper [BeGiCy05b] explores two ways for detecting timing channels with binary code and with multi-symbols code. The latter assumes that a skillful attacker will try to maximise the Shannon capacity of the covert channel and try to identify such a behaviour with probabilities. This assumption is weak: either the attacker is not skillful and he will not try to maximize the capacity of the channel or the attacker is really skillful, he knows the technique, and he will intentionally not use it to avoid detection.

The former assumes that the binary code uses only two different delays to code information. We can draw a graph with the number of packets (Y axis) given the transmission delay (X axis). If there is a covert channel in the network there will be two picks on the graph: one corresponding to the delay encoding one, the other to the delay encoding zero. There will be few packets with others timing values. In contrary, if there is no covert channel the repartition of packets will be concentrated on one pick and others timing values will be centred around.

Packet sorting techniques can be easily defeated if enough data is transmitted: too much entropy in the packets of a network is abnormal.

Detection requires a certain amount of data, thus a covert channel that exfiltrates just a few bytes to transmit a password or a private key will likely not be detected.

2.3 Process Query Systems

An overview of Process Query Systems (PQS) is available in [PQS04]. PQS are a new retrieval information technology where queries are described as process descriptions. Given a set of events the user can identify with a PQS which processes has generated which events.

A PQS is composed of four elements:

- a stream of observable events.
- a set of models that describes potential processes that can generate the events.
- some tracking algorithms that analyse events to find which processes have generated them.
- a PQS engine that take the three previous elements in input and give in output the models likelihoods for the observed events.

Models, which represents dynamical systems, can be represented with different formalisms: with Hidden Markov Chains, Finite State Machines with hidden states, Hidden Petri Networks etc. Tracking algorithms rely on Viterbi or Viterbi-like algorithm.

We should emphasize the fact that Process Query Systems have a broad range of applications. They can been used for instance to analyse social networks and discover terror cells or business processes, to track fish or humans on videos and, in our case, to monitor networks.

PQS are used in [BeGiCy05] to detect timing covert channels with binary codes. The authors indicate that some PQS models can return false positives, that is some processes are identified as covert channels when they are not. They do not provide enough data to see how successful their method is with other models but a PSQ system which includes Snort probes, can be downloaded and tried on their website: http://pqsnet.net.

We should emphasize the fact that using a PQS system for detecting a timing covert channels with binary code seems excessive.

2.4 Neural networks

Neural networks (NNs) try to imitate the behaviour of our brain. A neural network is constituted by neurons and connections between them. Input neurons receive information, called signals, hidden neurons contribute to reasoning and output neurons are used to retrieve the result of the process. Each connection between two neurons has a weight. A neuron takes the sum of its inputs, balanced by each weight, applies a function to it and outputs a corresponding signal. The function can be, for instance, a sigmoid function. This simple architecture reproduces some aspect of our neurons and is enough to build a system with some learning capabilities.

Neural networks can be used to approximate functions, and for patterns or sequences recognition. NNs can be trained to recognize a given input, in our case to recognize TCP/IP characteristics used by covert channels. During the training a set of inputs is transmitted to the NN and the weight of the connections are adjusted to match a given output. NNs are powerful: they can generalize, i.e., give meaningful outputs even if they did not encounter the inputs during training. A drawback of neural networks is the time needed to train the network, which can be long.

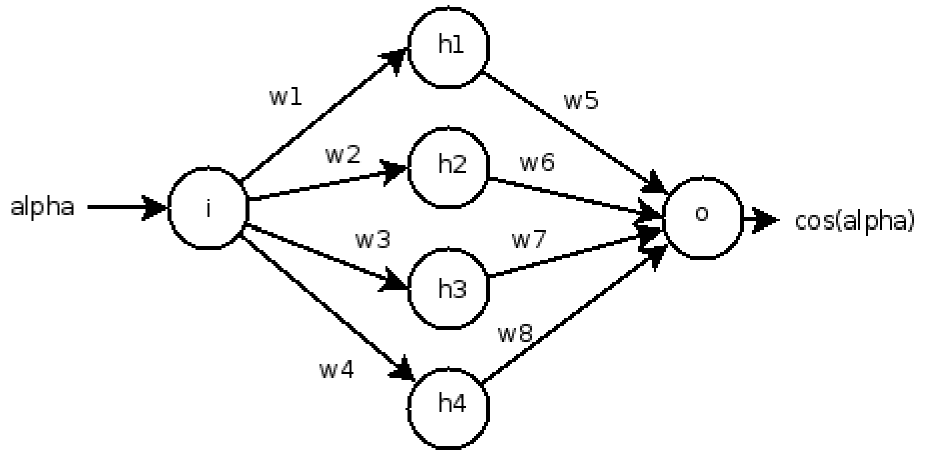


Illustration 1: A simple neural network with an input neuron and an output neuron. With a sigmoid function for neurons and appropriate weights, this simple network is able to give a good approximation of the cosinus function.

NNs are used in [TuAn05] to detect NUSHU. Several NNs are first trained to recognize TCP ISNs generated by NUSHU on an example set of TCP segments. Once the training is done, real segments are transmitted and the NNs try to identify the segments of the covert channel. ISNs generated by NUSHU do not covert the same space as a classic Linux TCP/IP stack; that is why the NNs can differentiate between ISNs generated by NUSHU and others. The NN is recurrent, thus taking into account ISNs already seen to interpret the ISN being analyzed.

The best NN has a false channel detection rate inferior to 0.1%. The accuracy of results begins to be good for fifty ISNs examined or more.

NNs are not limited to the detection of a particular type of covert channels and could be used to detect a lot of covert channels but a simple or statistical analysis is often enough. Drawbacks are the time needed to train the networks, the difficulty to find the optimal number of neurons, and the existence of false positives.

2.5 Support vector machines

Support vector machines (SVM) are, in a sense, similar to NNs. They are learning machines used to identify patterns but instead of trying to imitate the brain like NNs they use mathematical and optimization techniques. The term “machine” should not confuse the reader: a SVM is not physical but just a mathematical concept. Like NNs support vector machines need to be trained before doing classifications. Support vector machines do classification on vectors.

Let us take a really simple example to illustrate support vector machines. In the following figure red and blue vectors of two dimensions are represented:

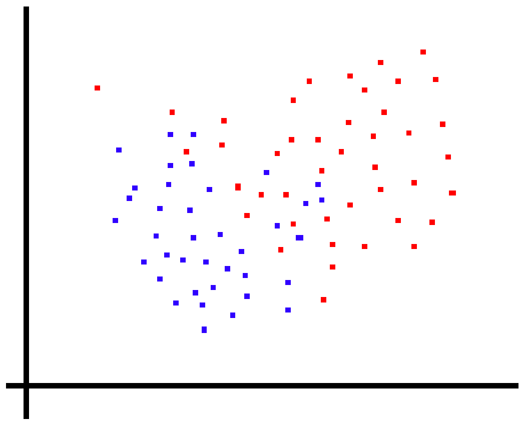


Illustration 2: Two classes of vectors of dimension 2.

We want to classify blue and red vectors. In this case they are not linearly separable. A SVM considers the problem in the higher dimension, and finds an hyperplane to separate the vectors. The same principle apply to vectors of dimension n. The classification problem is seen as a quadratic optimization problem by the vector machine.

[SoSeMo03] demonstrates the use of support vector machines to identify covert channels. The authors try to identify two types of covert channels: covert channels that use the identification field of IPv4 and the ones that use the ISN field without encryption. Both techniques just encode ASCII values into the ISN with simple transformations that generate a more realistic number.

The authors propose a detection method based on the time relation between the packets and the modified field in packets. This is a similar approach to the NNs seen in the section before that use packets already seen in addition to packets being seen to identify a covert channel. The training of the SVM is done on 10 000 packets. Results on the test sets are good, with a detection rate approaching the 99%. It is not a surprise: the covert channels used for the test are not really stealth and could be identified with simple or statistical (randomness) analysis.

2.6 Undetectable covert channels

A major work in the field of covert channels is the work done in [MuLe05]. The authors study the algorithms used to generate ISNs in OpenBSD and Linux. OpenBSD uses RC4 to assure randomness of ISNs while Linux uses MD4. The authors use this cryptographic algorithms to encode the data of their covert channels in the ISNs and encode them in a way identical to ISNs generated by OpenBSD or Linux. Thus the ISNs generated by the covert channels cover the same space as the ones generated by a system without covert channels. There is no public implementation of the algorithms but a correct implementation can create a covert channel totally undetectable and with a content that can be only be read with the private key. The only drawback with such a covert channel is that the covert channel has a very low bandwidth: only a few bits per connection.

3 Avoiding covert channels

We have seen the existing methods to detect covert channels. These methods should be used as much as possible to discover covert channels in networks. In addition techniques to avoid certain covert channels can be used. In this section we consider techniques with traffic normalizers and how to secure machines to avoid covert channels.

3.1 Traffic normalizers

A traffic normalizer filters and modifies the traffic between machines, and thus making the elaboration of covert channels more complex. Traffic normalizers are presented in [HaPa01] and in [FiFiPaNe03] with the name of “active wardens”. In both cases the aim is to remove the ambiguities in the protocol and to make some modifications to counter covert channels.

Part of the work consists of enforcing the TCP and IP norms: fields that should be to zero, like reserved flags or urgent pointers when the URG flag is not set, are set to zero by the normalizer. All ambiguities that can be suppressed without breaking the semantics are suppressed. Illegal packets, for instance packets with an invalid source address, can be rejected.

The other part of the work is a deeper modification of the headers to avoid covert channels. IP identification fields are set to zero when there is no fragmentation. When there is fragmentation the IP identification of fragments are changed and randomized.

The problem with covert channels that use ISNs is more complex. [FiFiPaNe03] proposes to add an offset to the ISNs. Clearly this will change the message received. But the sequence number field is only 32 bits: we can put a marker in the covert message and brute force the received message with different offsets to retrieve the original marker. We then obtain the offset used by the normalizer and we can retrieve the original message in its whole.

For timing covert channels, random delays can be added on each hops in the path. The covert channel will be less reliable.

Normalizers can be used in conjunction with detection mechanisms: the detection mechanism can discover that a covert channel exists, while the normalizer reduces the possibility of data exfiltration. Of course, packets for the detection must be captured before the normalization. If the covert channel's communication is bidirectional a normalizer can break it, making its detection more complex. Using normalizers, detection mechanisms, or both depends on the security policy and on what is more important: data protection or intrusion detection.

In practice traffic normalizer tools are almost non-existent but they could be implemented as part of firewalls or routers.

3.2 Security of machines

When legitimate users do not try to set up covert channels themselves, covert channels are due to an attacker trying to communicate with a compromised machine. Increasing the security of machines reduces the risk of covert channels since covert channels typically require administrative privileges.

The security of machines is a too broad subject to be addressed exhaustively in this paper but we give a minimal list of hints to apply to reduce the risk of attacks leading to covert channels and data exfiltration.

- Anti-virus, anti-rootkits, and firewalls are a minimum for workstations.
- Only digitally signed software and software from trusted sources must be executed.
- Cryptography can be used to encrypt important data and restrict their exfiltration.
- Systems and software must be kept up to date.
- Software that makes stack overflows and heap overflows harder to exploit must be used.
- Host intrusion detection system (HIDS) and host integrity checkers must be used.
- Users must be made aware of security concerns.
- Physical access to machines and to the network must be restricted to authorized users.
- A security policy must be set and enforced.

This list is endless but applying these advices is already a good step to secure machines. An important step to secure machines is to apply the principle of least privilege. Each application, operating system service, and user must only have access to what it needs to perform its work. Applications must be contained so that a compromised application can not put at risk on other applications or the system itself.

Modern operating systems such as Linux with SELinux activated, or to a far lesser extent Windows Vista, possess such containment features.

3.3 Lowering the potential of covert channels

As we have seen, detecting covert channels is not always possible. Detecting and stopping covert channels before data exfiltration is even harder since we need exfiltrated data of the covert channel to detect it: we can only stop the communication once some data have been exfiltrated. This last assumption holds when the first IP packet is not enough to detect the covert channel. If a covert channel can be detected with only one packet, we can delay the establishment of the communication to examine the packet, and block the communication if its a malicious one.

Moreover, few applications exist to detect covert channels and most enterprises do not try to identify if there is a covert channel or not in their network. Setting up good security measures on machines to avoid covert channels can help but as we all know, a skilled attacker can always find a new and unpatched vulnerability.

With these facts in mind we can assert that totally stopping covert channels is impossible. If covert channels can not be stopped, we need to decrease their malicious potentials, especially in an environment with strong security requirements. The most effective method is to reduce the bandwidth or to close the channel. For instance enterprise LANs can be closed outside working hours. General advice for dealing with covert channels can be found in [Hu95].

Conclusion

We have presented the TCP and IP protocols and the covert channels techniques related to it. The IPv6 protocol, which will replace IPv4, is also vulnerable to covert channels.

Simple detection methods have been studied as well as complex detection methods like PQS, NNs and SVM. Complex methods can have good detection rates but they require training and are subject to false positives. There are other techniques that do not require learning techniques to detect covert channels. Simple behaviour analysis or statistical analysis can detect most of the covert channels. Despite that, few applications implement covert channel detection. It is not a common practice in enterprise networks, due to a lack of knowledge and a lack of standard detection solutions. There is also a lack of tools for traffic normalization, which prevent from several covert channels.

Covert channels that transmit only a few bytes can be hard to detect and a technique exists to set up a totally undetectable covert channel. The possibility of an undetectable covert channel reminds us of the real threat of covert channels. We have seen that improving security of machines reduces the covert channels risk and we must always keep in mind that covert channels should been seen in the context of security as a whole to minimize their risks.

Bibliography

[**Aba01**] C. Abad, *IP Checksum Covert Channels and Selected Hash Collision*, 2003

[**Aba05**] C. Abad, Internet Checksum Covert Channels, http://www.the-mathclub.net/index.php/Internet_Checksum_Covert_Channels

[**Ash02**] Kamran Ahsan, *Covert Channel Analysis and Data Hiding in TCP/IP*, 2002

[**BeGiCy05**] Vincent Berk, Annarita Giani, George Cybenko, *Covert Channel Detection Using Process Query Systems*, 2005

[**BeGiCy05b**] Vincent Berk, Annarita Giani, George Cybenko, *Detection of Covert Channel Encoding in Network Packet Delays*, 2005

[**CaBrSh04**] Serdar Cabuk, Carla E. Brodley, and Clay Shields, *IP Covert Timing Channels: Design and Detection*, 2004

[**FiFiPaNe03**] Gina Fisk, Mike Fisk, Christos Papadopoulos, and Josh Neil, *Eliminating Steganography in Internet Traffic with Active Wardens*, 2003

[**GiGrLiTi02**] John Giffin, Rachel Greenstadt, Peter Litwack, Richard Tibbetts, *Covert Messaging Through TCP Timestamps*, 2002

[**Gra94**] James W. Gray, *Countermeasures and Tradeoffs for a Class of Covert Timing Channels*, 1994

[**HaPa01**] Mark Handley and Vern Paxson, *Network Intrusion Detection: Evasion, Traffic Normalization, and End-to-End Protocol Semantics*, 2001

[**Hu95**] McHugh, J., Naval Research Laboratory, *Handbook for the Computer Security Certification of Trusted Systems*, 1995

[**LuLeCh06**] Norka B. Lucena, Grzegorz Lewandowski, and Steve J. Chapin, *Covert Channels in IPv6*, 2006

[**MuLe05**] Steven J. Murdoch and Stephen Lewis, *Embedding Covert Channels into TCP/IP*,

[**NCSC TCSEC**] National Computer Security Center, Department of Defense Trusted Computer, *System Evaluation Criteria, DoD 5200.28-STD*, 1985

[**PQS04**] George Cybenko, Vincent Berk, Valentino Crespi, Robert S. Gray, Guofei Jiang, *An Overview of Process Query Systems*, 2004

[**RFC2460**] 1998, Internet Protocol, Version 6 (IPv6), http://www.ietf.org/rfc/rfc2460.txt

[**RFC791**] RFC 791, Internet Protocol, http://tools.ietf.org/html/rfc791

[**RFC793**] RFC 793, Transmission Control Protocol, http://tools.ietf.org/html/rfc793

[**Row96**] C. H. Rowland, Covert channels in the TCP/IP protocol suite, http://www.firstmonday.org/issues/issue2_5/rowland/

[**Rut04**] Joanna Rutkowska, *The implementation of passive covert channels in the Linux kernel*, 2004

[**SoSeMo03**] Taeshik Sohn, JungTaek Seo, and Jongsub Moon, *A Study on the Covert Channel Detection of TCP/IP Header Using Support Vector Machine*, 2003

[**Star02**] P. Starzetz, Ambiguities in TCP/IP, http://gray-world.net/papers/ambiguitiesintcpip.txt

[**Ste93**] W. Richard Stevens, *TCP/IP Illustrated, Volume 1*, 1993

[**TuAn05**] Eugene Tumoian, Maxim Anikeev, *Detecting NUSHU Covert Channels Using Neural Networks*, 2005

[**ZaArBr06**] Sebastian Zander, Grenville Armitage, Philip Branch, *Covert Channels in the IP Time To Live Field*, 2006