11-2005

A Covert Channel in Packet Switching Data Networks

Bo Yuan
Rochester Institute of Technology

Peter Lutz
Rochester Institute of Technology

Recommended Citation

B. Yuan and P. Lutz, "A Covert channel in packet switching data networks," Proceedings of The Second Upstate New York Workshop on Communications and Networking, 2005, Rochester, New York

A Covert Channel in Packet Switching Data Networks

Bo Yuan and Peter Lutz
Department of Networking, Security, and Systems Administration
Golisano College of Computing and Information Sciences
Rochester Institute of Technology
Rochester, New York 14623
{bo.yuan, peter.lutz}@rit.edu

Abstract— **This paper presents a covert communication channel that exists in virtually all forms of packet switching data networks. On the one hand, this covert channel, if used properly, can potentially enhance the overall security of data communications over networks. On the other hand, the covert channel can also potentially become a back door to access a destination computer, and hence becomes a security hazard to the computer. A simple protocol is specified for communications on the covert channel. A modified TFTP application is also presented to demonstrate how to use the covert channel to convey secret messages or to enhance the integrity of data communications. The application also illustrates a back door that leaks client's data files without user notification. A sliding entropy method is also introduced to detect some cases of covert channels.**

I. Introduction

As pointed out by Jonathan Millen in [1], the term ”covert channels” was first introduced by Lampson in 1973 [2]. Since then, it has evolved into many different, yet very similar meanings. Gilgor listed five definitions of covert channels in [3]. In the authors' view the original definition by Lampson in [2] is the most general and the most applicable to broad areas of communications. That is, a communication channel is *covert* if it is neither designed nor intended to transfer information at all.

At its simplest form, covert channels are used to describe communication between system processes and/or storage devices. Its uses have been broadened to other forms of communication, most recently in computer networks.

In [4], Rowland outlined three methods to manipulate TCP/IP header information so as to form covert channels between a source and a destination. The first method is the manipulation of the IP identification (ID) field. The ID field is intended to label an IP frame. When fragmentation occurs at the data link layer, the ID field is used to identify the fragment frames so that they may be reassembled back to the original, unfragmented IP frame. The ID field is a two-byte field. Thus, instead of using an arbitrary number, an ASCII code may be used to send a short message.

The second method is to use the initial sequence number field. When a TCP connection is established, a ”three way handshake” of SYN/ACK packets occurs. One of the purposes of this handshake is to negotiate the starting value of the sequence numbers to be used for packets traveling in either direction. The initial sequence number used in SYN/ACK packets is randomly generated. Rowland proposes to use an ASCII code multiplied by 16777216 instead of using random numbers for the initial sequence numbers. Thus, short messages may be embedded in the apparently innocuous choice of sequence numbers.

The third method is to use forged source and destination IP addresses to bounce from an intermediate site to a destination. This involves three hosts, the sender (S), an intermediate, innocent host called the bounce server (B), and the intended destination host (D). The data is encoded in the initial sequence number field (ISN), as above. The SYN packet then is sent by S to B. However, the destination IP address is spoofed to appear to be that of D. When B receives the SYN packet, it responds to D (not S) with the original ISN +1. When D receives the reply, it can simply subtract 1 from the ISN and receive the data. This hides the actual sender (S) of the information from observers as well as from D.

In [5], Giffin, *et al* extended Rowland's work. They discussed using the TCP timestamp option field to exchange information. In [6], Bauer presented several covert channels via HTTP servers. This previous work focuses on a specific protocol and encodes information in one or more fields in that protocol.

This paper presents a more generic covert channel based on a very basic property of packet switching and, hence the covert channel exists on almost all packet switching data networks. It can be implemented at different layers of the network, regardless of the platform involved. A simple protocol is defined to facilitate communication on the covert channel. A demo system is also presented to illustrate possible uses of the covert channel.

II. A Covert Channel in Packet Switching Networks

In a packet switching network, a message is divided into many packets of various sizes that are usually much smaller than the entire message. Each packet is then transmitted individually into the network. Packets may follow different routes to their destination. Once all the packets arrive at the destination, they are reconstructed back into the original message. Packet switching is the foundation of our today's data communication networks. The entire Internet is a large packet switching data network. Packet switching has also been extended into the areas where circuit switching was traditionally used. For instance, voice over IP based Internet phones are gradually replacing analog telephone systems.

However, there is one issue related the packet switching that so far has been overlooked. This issue bears on the very nature of the packet switching itself. When a message is divided, packets can be sent with different sizes. Only maximum packet sizes are usually specified in a protocol so that packets can travel through networks of different technologies or models. For example, the largest Ethernet II frame is 1526 bytes including the preamble [7]. The fact that packets can be different sizes provides us another way to exchange information between a source and a destination.

Here is an example of how this can be achieved. Suppose we need to send a letter "a" from a source to a destination. The ASCII code of letter "a" is 1100001. Select an arbitrary message with a sufficient size. We then divide the message into packets whose sizes follow this pattern: odd, even, even, even, even, odd, odd. A packet with an odd size represents "1", and an even size represents "0". When packets reach the destination, the destination computer checks the size of each packet and converts packet sizes back to 0 or 1. In this case, it is odd, even, even, even, even, odd and odd, which is 1100001 (read from lowest order bit to highest), i.e., the letter "a" in ASCII.

The information, the letter "a", is transmitted by the packets from the source to the destination. This communication channel is referred to as the *the covert channel*, or the *hidden channel*, or the *second channel*, of the packet switching data network. The message reconstructed from the packets is the main message, and its communication channel is referred to as the *main channel* of the packet switching data network.

The potential for variable sized of packets exist in all packet switching network technologies such as Ethernet, X.25 and frame relay (except ATM, which uses fixed sized cells). In an X.25 network, the network protocol allows the user send packets up to 128 bytes long [8]. With frame relay technology, the user can send frames of up to 1600 bytes between the two ends of a leased permanent virtual circuit [8]. Payload sizes of Ethernet frames are between 46 and 1500. But more importantly, variable sized packets are allowed by many protocols at higher layers of OSI model. At the data link layer, the PPP protocol allows payload sizes up to 1500 bytes; at the network layer, both IP and IPX protocols have a 2 byte packet length field in their data format, which can specify packet sizes of up to 65536 bytes; at the transport layer, UDP specifies the packet length with 2 bytes; TCP uses its sequence numbers to indicates different amounts of data (*i.e.*, the sequence numbers are incremented by the size of the payload with each packet). Thus, the covert channel potentially lives at every corner of today's data networks.

III. The Capacity of the Covert Channel

The capacity of the covert channel is defined as the amount of information it carries relative to the main channel. This depends, in part, on what protocol is used on the main channel. For example, if a modified TFTP protocol is used as demonstrated in the later section, the maximum data transmission is 512 bytes per packet. Using the binary encoding, i.e., one packet on the main channel represents one bit on the covert channel, the minimum capacity ratio is 1:4096. If ASCII encoding is used, the minimum ratio is 1:512. In general, the capacity ratio can be calculated by the formula

$$\frac{log_2 L}{M} \tag{1}$$

where M is the maximum size of a user data unit in bits per packet; and L is the number of different sizes that packets are allowed to use to encode information on the covert channel.

A user data unit of a packet is the application data unit of the packet after all header or trailer information at different layers is removed, i.e., it is the actual application data transmitted by a packet. Thus, the maximum size of a user data unit is neither the maximum size of the protocol data unit (PDU), nor the maximum size of the transmission unit (MTU).

According to [7] p. 36, a host is not required to receive a datagram larger than 576 bytes. Thus many UDP based applications limit themselves to 512 bytes for user data. Thus, 1:4096 is the minimum capacity ratio of the covert channel to the main channel in most cases of UDP based applications. For TCP based applications, the maximum segment size (MSS) is negotiated between the sender and the receiver before transmission. Its value is selected by both sides to ensure no fragmentation occurs on the transmission path from the source to the destination. The default MSS is 536 bytes. For Ethernet, its value can be up to 1460 bytes before fragmentation. Thus, in this case, the minimum capacity ratio between the covert channel and the main channel is 1:11680.

It seems that this ratio is disappointingly small. But considering massive network traffic on the Internet today at any given moment, the covert channel, even with its minimum capacity ratio, would be able to transmit a very large amount of data. Furthermore, the covert channel need not be used to transmit massive amounts of data in the first place. It can be used to transmit keys for authentication, or short secret messages. It can also transmit some small but critical information to enable understanding the data on the main channel, or control information related to data connections, etc. The potential of the covert channel is limited only by our imagination. We foresee many applications based on the covert channel in near future.

IV. Applications of the Covert Channel

The covert channel can be used to transmit hidden messages. It is not a real data communication channel. It lives "under" the main channel. The "medium" of the covert channel is packets that carry data on the main channel. No additional packets are generated. Variation of packet sizes is normal behavior in packet switching data networks. Thus, traffic on the main channel is not noticeably different from the other traffic. Hence the covert channel is invisible and difficult to detect. An eavesdropper has to recognize the traffic and has to know the way to interpret the traffic pattern in order to intercept the message on the covert channel. Although the capacity of the covert channel is much smaller than the main channel, it is proportional. The larger the message on the main channel, the more information carried on the covert channel. A more extended discussion on covert channels and data hiding can be found in [9].

The covert channel can be used together with the main channel to enhance the security of the main channel. For example, the message transmitted on the covert channel can be employed as the integrity check for the message on the main channel. The sizes of packets transmitted on the main channel can be designed to follow some predefined pattern. If a packet is modified during transmission, the pattern may be broken at the destination. The modification attack, hence, can be detected by the receiver.

As an example of the use of this technology, a covert channel-aware ftp client can download a file and automatically verify the MD5 checksum of the file. When the server sends a file, packets are delivered in a pattern of sizes that represents the MD5 checksum of the file. As soon as the download completes, the ftp client recalculates the checksum and compares it with the one received via the covert channel. The current, traditional approach requires two downloads to complete the process. The checksum of a file is downloaded separately with the file. A problem with this approach is that the downloaded checksum is also subject to modification. This method requires just one download. It would be much more difficult for an attacker to construct a modification to the packets during transmission which also maintains the matching MD5 checksum patten on the covert channel. A typical 128 bit MD5 checksum just needs 128 packets to complete transmitting the checksum at the worst case, using binary encoding; 16 packets to complete using ASCII encoding.

V. Risks

The covert channel can potentially be a security risk to the destination computers. It can serve as a delivery channel for viruses, worms, or other types of malware. When a covert channel-aware application is installed onto a user's computer, a back door is opened. Without reviewing its source code, it is very difficult to identify if software is covert channel-aware or not. (Note that this may be a strong argument for open source software.) Such software usually performs all functions that it expected to do and it interprets covert messages only when there are messages on the covert channel. There are no extra network connections needed to download data via the covert channel. Thus, network traffic does not appear different from 'normal' traffic, even though the covert channel is in use. Neither today's profile based, nor protocol based network defenses will be able to detect any anomalies. Furthermore, an attacker can select when to feed data on the covert channel and what data to send; hence, the client computer may be under the complete control of the a remote attacker.

One possibility for defending against this weakness of the packet switching networks is to unify packet sizes at all layers of the OSI model and for all protocols. This would seem a very difficult goal to achieve, at least from today's perspectives. Another way to detect such covert channels is to monitor variations in packet sizes, which is the topic of the next section.

VI. Sliding Window Entropy Detection Method

To detect variations in packet sizes, a nature method to use is to measure the entropy of packet sizes. Since it just needs a small number of packets of different sizes to convey a secret such as a key or a password on the covert channel, a majority of packet may be transmitted with a constant size. Thus, the entropy of packet sizes for an entire transmission may be biased by the number of constant size packets and may be too small to alert any irregularity. One suggestion is to use the sliding window approach as described as follows.

Suppose $p_1, p_2, ..., p_n$ is a sequence of n packets. l_i denotes the size of packet p_i for $i = 1, 2, ..., n$. For a given integer, m, called *window size*, E_i is the Shannon entropy of the probability distribution of packet sizes $\{l_i, l_{i+1}, ..., l_{i+m-1}\}$ within the window, i.e., from p_i to p_{i+m-1}, for $i = 1, 2, ..., n - m + 1$. Then, the maximum value of E_i is log_2^m, when all packets in the window have different sizes; and the minimum value is 0, when all packets have the same size. The general Shannon entropy of a window can be calculated by the following formula.

$$E_i = log_2^m - \frac{1}{m} \sum n_j log_2^{n_j} \tag{2}$$

where m is the window size; n_j are counts of different packet sizes within the window.

The size of the window is a variable set by the user. The smaller the window size, entropy values are more sensitive to packet size and the smaller variations can be detected; the wider the window size, the less sensitive to the variations of packet sizes.

Note that when high entropies are detected in packet sizes in a traffic, it is not necessary the case that a covert channel is in use. Some types of network traffic are inherently with high entropies in packet sizes. For instance, traffics of online chat dialogs or instant messaging have this characteristics due to different lengths of text messages.

For efficiency, most applications maximize packet sizes allowed by a protocol during transmission. Thus, constant zero entropies should be detected in most time. Any non zero sliding entropies indicate that a hidden channel may be in use.

VII. A Simple Protocol for the Covert Channel

To facilitate communications on the covert channel, a protocol is needed. Both the encoding method, the message length and the message type should be specified. Figure 1 illustrates an example of a data format for communication on the covert channel.

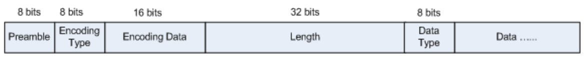


Fig. 1. Covert Channel Data Format

Preamble is a byte of alternating 0s and 1s that indicates the beginning of a data transmission on the covert channel. The pattern should be 10101011. When it is transmitted, bit 1 can be represented by either even or odd packet sizes. The receiver should determine which representation is in use ones from the pattern of the preamble. For instance, if the pattern ends with two packets of odd sizes, packets with odd sizes represent bit one. Note that, while 8 bytes of preamble is used in Ethernet, only one byte is recommended here due to the limited capacity of the covert channel.

Encoding type is also one byte. It indicates what types of encoding will be used in the subsequent communication. Value 175 or 10101111 in this field represents binary encoding. That is, one packet represents one bit. Value 170 or 10101010 in binary, indicates the ASCII encoding. That is, packet sizes represent ASCII codes, so each packet represents 8 bits.

Encoding data is two bytes. Its values have different meanings depending on the encoding type. When the encoding type is ASCII, its value defines the following mapping from packet sizes to ASCII code.

$$C = min(S(p) - T, 127) \tag{3}$$

where $S(p)$ is the size of the packet p; and T is the decimal value specified in the encoding data field; and C is the decimal value of an ASCII code represented by the packet p. In the case of binary encoding, this field is meaningless.

Length field of four bytes indicates the total number of packets that will be used to transmit data on the covert channel.

Data type is one byte. It indicates the meaning of the data so that the receiver can interpret it. For example, when data type is 99 in decimal, the data is the checksum of the data transmitted on the main channel; when it is 88, the data is a secret message; when it is 77, the data is a file name with full path on the client's machine; when it is 66, the data is the contents of the file that was specified in the previous communication.

Data field contains packets that convey information on the covert channel. The length of the data field is specified in the *length field*.

Note that the first four bytes in the data format specified in Figure 1 are always transmitted by binary encoding. The rest of bytes are transmitted by the encoding method specified by the two fields, encoding type and encoding data.

This simple protocol is designed for our demo system only. More study is needed to improve the protocol. Note that applications of the covert channel with malicious intention may not follow any protocols. They may design their own protocols to further encrypt messages carried on the covert channel.

VIII. An Implementation of the Covert Channel Based on a Modified TFTP Protocol

The UDP protocol by J. Postel [10] in the RFC 768 specifies the following packet format.

```
 0      7 8     15 16    23 24    31
+--------+--------+--------+--------+
|     Source      |   Destination   |
|      Port       |      Port       |
+--------+--------+--------+--------+
|                 |                 |
|     Length      |    Checksum     |
+--------+--------+--------+--------+
|
|          data octets ...
+---------------- ...

   User Datagram Header Format
```

The length field in the UDP header is the length in octets of this user datagram including this header and the data. The length of all other fields are fixed except the data field. Thus, we can control the length field by specifying how many octets to be put in the data field so as to demonstrate the covert channel at the network layer. UDP is selected due to its simplicity and ease of implementation.

First we make a small change to the Trivial File Transfer Protocol (TFTP) [11], a UDP based protocol. In fact, the only thing we modify is the handling of the end-of-file condition. In TFTP, a packet with a data size less than 512 byte indicates the end of transmission. If the last packet is 512 bytes long, another packet with zero data size needs to be transmitted. In our modified TFTP (MTFTP) protocol, only a packet with a zero data size can terminate a transmission.

A screen shot of the MTFTP server is illustrated in Figure 2. In addition to functions such as downloading and uploading files from/to the server, the MTFTP server also has three additional functions enabled through the covert channel. The first function is to send a secret message to the client through the covert channel. When the message button is checked, a user at the server side can type in a secret message to send to a client.

The second function is the checksum function. When it is checked, the server calculates the checksum of a file being transmitted on the main channel and sends the checksum through the covert channel to the client simultaneously. As soon as the client receives the file, it also has the checksum of the file on the covert channel. The client can then compare the checksum received and the checksum calculated of a file and decide to accept or reject the file.

The third function is called ”GET FILE.” When this button is checked, the user can specify a file name with full path at the server side. When a client downloads a file from the server, this ”GET FILE” command and the file name are transmitted to the client without the human user's knowledge. The next time the client uploads a file to the server, the client program also sends the contents of the chosen file to the server via the covert channel, again without the knowledge of the human user of the client application. This function demonstrates a back door to access the client machine via the covert channel.

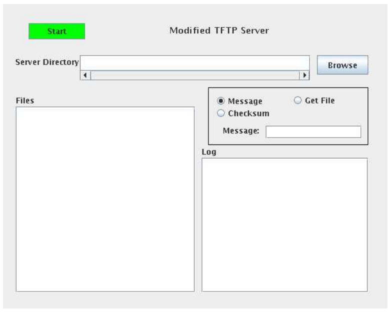


Fig. 2. Modified TFTP Server

Besides downloading and uploading files, the modified TFTP client is also able to understand messages on the covert channel. When the data type is 99 on the covert channel, the client compares the checksum received from the covert channel and the checksum that the client calculates from the downloaded file. If they are the same, the download was successful; otherwise the download failed. When the data type is 77, the client will read the contents of the specified file into memory; it will then send those contents back to the server via the covert channel. These extract functions are performed behind the scenes; the human user may not know about these activities. Figure 3 is a screen shot of the demo client.

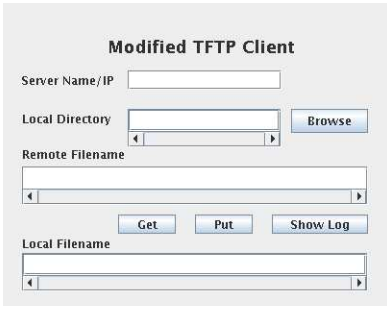
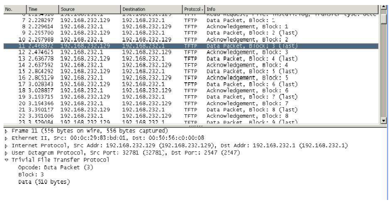


Fig. 3. Modified TFTP Client

From the network packet captures, we can see that the TFTP packets have varying sizes. The only noticeable behavior is that TFTP packets with a size less than 512 are not end of the transmission, as specified in [11]. See Figure 4. A sliding entropy method is proposed to detect covert channels. This is mainly due to the capture software recognizing these packets as TFTP packets.

Fig. 5 shows sliding entropies of variations in packet sizes when the covert channel is used to send a short message with the window size 10 and the step size 1. As you can see that after packet 63, packet size becomes a constant, which is the normal characteristic of a TFTP transmission.



```
No.  Time      Source           Destination      Protocol  Info
  7 2.228297   192.168.232.129  192.168.232.1    TFTP      Data Packet, Block: 1
  8 2.229614   192.168.232.1    192.168.232.129  TFTP      Acknowledgement, Block: 1
  9 2.295700   192.168.232.129  192.168.232.1    TFTP      Data Packet, Block: 2 (last)
 10 2.297988   192.168.232.1    192.168.232.129  TFTP      Acknowledgement, Block: 2
 11 2.468872   192.168.232.129  192.168.232.1    TFTP      Data Packet, Block: 3 (last)
 12 2.474625   192.168.232.1    192.168.232.129  TFTP      Acknowledgement, Block: 3
 13 2.636778   192.168.232.129  192.168.232.1    TFTP      Data Packet, Block: 4 (last)
 14 2.637592   192.168.232.1    192.168.232.129  TFTP      Acknowledgement, Block: 4
 15 2.804292   192.168.232.129  192.168.232.1    TFTP      Data Packet, Block: 5 (last)
 16 2.805129   192.168.232.1    192.168.232.129  TFTP      Acknowledgement, Block: 5
 17 3.028343   192.168.232.129  192.168.232.1    TFTP      Data Packet, Block: 6 (last)
 18 3.028837   192.168.232.1    192.168.232.129  TFTP      Acknowledgement, Block: 6
 19 3.193715   192.168.232.129  192.168.232.1    TFTP      Data Packet, Block: 7 (last)
 20 3.194366   192.168.232.1    192.168.232.129  TFTP      Acknowledgement, Block: 7
 21 3.390157   192.168.232.129  192.168.232.1    TFTP      Data Packet, Block: 8 (last)
 22 3.391006   192.168.232.1    192.168.232.129  TFTP      Acknowledgement, Block: 8
 23 3.529084   192.168.232.129  192.168.232.1    TFTP      Data Packet, Block: 9 (last)

▷ Frame 11 (556 bytes on wire, 556 bytes captured)
▷ Ethernet II, Src: 00:0c:29:83:bd:01, Dst: 00:50:56:c0:00:08
▷ Internet Protocol, Src Addr: 192.168.232.129 (192.168.232.129), Dst Addr: 192.168.232.1 (192.168.232.1)
▷ User Datagram Protocol, Src Port: 32781 (32781), Dst Port: 2547 (2547)
▽ Trivial File Transfer Protocol
    Opcode: Data Packet (3)
    Block: 3
    Data (510 bytes)
```

Fig. 4. An Ethereal Capture File

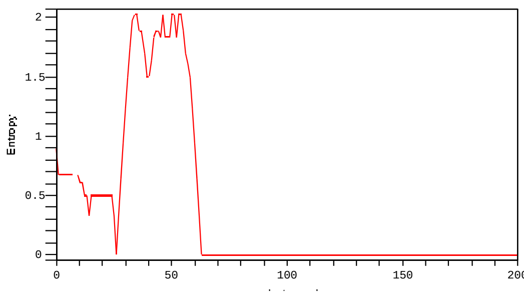


Fig. 5. Sliding Entropies

The source code of the demo system can be obtained by contacting the first author.

IX. Conclusions

The paper presents a covert channel existed in packet switching data networks. It is demonstrated that the covert channel can be used to convey secret messages. It can be used to enhance the integrity of data communication on the main channel and can also be a back door to leak information from the client's host computer. A sliding entropy method is introduced to detect some cases of this kind of covert channels.

References

[1] J. K. Millen, "20 years of covert channel modeling and analysis." in *IEEE Symposium on Security and Privacy*, 1999, pp. 113–114.

[2] B. W. Lampson, "A note on the confinement problem," *Communications of the ACM*, vol. 16, no. 10, pp. 613–615, 1973.

[3] V. D. Gligor, "A guide to understanding covert channel," *National Computer Security Center*, 1993.

[4] C. H. Rowland, "Covert channels in the TCP/IP protocol suite." *First Monday*, vol. 2, no. 5, 1997.

[5] J. Giffin, R. Greenstadt, P. Litwack, and R. Tibbetts, "Covert messaging through TCP timestamps." in *Privacy Enhancing Technologies*, 2002, pp. 194–208.

[6] M. Bauer, "New covert channels in HTTP: adding unwitting web browsers to anonymity sets," in *WPES '03: Proceedings of the 2003 ACM workshop on Privacy in the electronic society*. ACM Press, 2003, pp. 72–78.

[7] W. R. Stevens, *TCP/IP Illustrated Volume 1: The Protocols*. New York: Addison-Wesley, 1994.

[8] A. S. Tandenbaum, *Computer Networks*, 3rd ed. Upper Saddle River, New Jersey: Prentice Hall PTR, 1996.

[9] M. Owens, "A discussion of covert channels and steganography," *SANS Institute: Information Security Reading Room*, 2002.

[10] J. Postel, "User datagram protocol," RFC 768, August 1980.

[11] K. Sollins, "The TFTP protocol (revision 2)," RFC 1350, July 1992.