A Covert Channel Over Transport Layer Source Ports

James R. F. Gimbi, Daryl Johnson, Peter Lutz, Bo Yuan
B. Thomas Golisano College of Computing & Information Sciences
Rochester Institute of Technology, Rochester, NY, USA

Abstract – *Covert communication is a rapidly expanding field of research with significant impact on the security theater. These communication methods, or "covert channels", can be applied in a number of ways, including as a mechanism for an attacker to leak data from a monitored system or network. This paper sets out to contribute to this field by introducing a new covert channel which operates over transport layer protocols. The mechanism is flexible, covert, and has the potential to operate at relatively high bandwidth. In addition, this paper proposes a number of encoding schemes which can be used in conjunction with this channel to improve its bandwidth and covertness.*

Keywords: Network Covert Channels, Information Hiding, Network Security

1 Introduction

A covert channel can be defined as any communication method where both the data being transmitted and the existence of the channel itself are hidden from network and system authority figures. The field has generated much interest because of its applications on both sides of the information security industry; while covert channels are useful for defensive security applications and collaboration between legitimate security teams, they can also be used by attackers to covertly leak data from a secure environment.

This paper presents a novel method for leveraging transport layer source ports as a medium for covert communication. The technique is flexible and can be applied in a wide variety of environments. The paper then discusses a number of possible implementations of this channel. It will also introduce a collection of encoding mechanisms to use in conjunction with the channel and will review their utility. Some of these encoding techniques provide data integrity and obfuscate the channel from a would-be investigator.

2 Related Work

Covert channels have been the subject of research for some time. They were originally defined in [1] as any communication medium not designed or intended for data transfer but could be used as such. Multiple types of covert channels have been defined, including storage channels, timing channels and behavioral channels. This topic was explored in depth in [2]. A storage channel is essentially any channel where a shared storage medium is used to encode and transmit information. A timing channel is any communication which relies on the time between particular events to encode and transmit information, instead of shared storage media. Behavioral channels are broadly defined as any channel where the mechanism is non-stored and time independent.

The channels covered in [1] are exclusively single system process-to-process examples. Since then, the definition of a covert channel has gradually expanded to include channels between processes on two separate machines over a computer network. [3] provided solid groundwork for creating TCP/IP timing channels. The approach encoded data in the amount of time between to the arrival of two packets. TCP/IP storage channels were thoroughly examined in [4]. In this work, data is transmitted in header fields of TCP/IP packets. Two known works have identified transport layer source ports as a potential channel medium in passing but did not discuss how it might be accomplished [5] [6].

3 Covert Channel Over Source Ports

This section defines and outlines a method for using transport layer source ports as a covert channel. A technical background will be provided in subsection 3.1. The method itself is introduced in 3.2. The final subsection will propose a number of different technical implementations.

3.1 Technical Background

Communication between two computers over modern network protocols requires the use of what is known as a network socket. A socket is a tuple of data used to identify each unique and active connection on a particular machine, and a socket pair is a tuple containing information for both the local and remote sockets [7]. While the exact contents of this tuple will vary depending on which transport protocol is being used, the TCP socket pair includes the IP addresses of both machines as well as the port numbers each machine has committed to the session. The IP addresses help the computer keep track of which remote machine it is communication with, while the port numbers help keep track of individual sessions for that machine. This 4-tuple allows two computers to manage thousands of unique conversations between them without risk of data loss on any one session. For example, a web client can make two distinct GET requests to a particular the web server for different page elements at port 80. Because they are two separate requests, different source port numbers are selected by the client (i.e. port 1,111 for one socket pair, and port 2,222 for the second socket pair) so the server knows which remote socket to feed the appropriate HTML response.

While most services are given a dedicated port number with which to manage all connections, client ports are not static and will not always be the same under normal operation. Instead, the source port is generally a pseudo-random number selected from a given range. Ports selected like this are known as ephemeral or temporary ports. This gives the client operating system flexibility when establishing new connections. There is technically nothing to prevent a client from using any port within the 16 bit port range (ports 0 through 65,535) but there are suggested standard ranges which most transport layer protocol implementations observe. A commonly observed range is maintained by the Internet Assigned Numbers Authority (IANA) which mandates that the two most significant bits must be registered as ones, giving a usable ephemeral range of 49,152 through 65,535 for a total of 16,384 available ports [8]. While Microsoft operating systems now follow the recommended IANA range, legacy Microsoft systems, such as Windows XP and 2000, use the range 1,025 through 5,000 for a total of 3,976 available ports [9]. Linux systems tend to vary from distribution to distribution but most use either the IANA mandate or the range 32,768 through 61,000 for a total of 28,233 available ports.

3.2 Transport Layer Covert Channel

The following method takes advantage of the flexibility provided by layer four protocols in source port selection and can be applied to any type of network environment that uses a layer four protocol, such as TCP and UDP. It consists of a sender, which will transmit data over the channel, and a receiver, which will collects the transmitted data. The sender and receiver must be able to communicate in some legitimate fashion without being flagged by a security appliance. For example, this pair might be a web server and client (TCP), a streaming media server and client (UDP), or some proprietary protocol.

Each time a new source port is needed there is an opportunity to transmit up to sixteen bits of information from the client to the server in that source port field. A one-way channel is established when a user or process manipulates the source port to send data. Because all that is modified for this channel is the contents of a mandatory static length field, it can easily be piggybacked on top of legitimate traffic. The channel lends itself to a large number of different encoding mechanisms, two of which will be outlined in the next section.

While it is possible to use these bits to transmit absolute data (i.e. sending an ASCII ‘A’ by using port 65), the channel is made more covert and robust by using the delta between two consecutive source ports. Using a delta scheme, no data is actually stored in a given source port; an analyst could investigate the totality of a guilty packet and find no leaked data. By contrast, absolute data transmission can easily be detected by an analyst reviewing a packet. Further, delta schemes lower the likelihood of colliding consecutive source ports because repeated characters will not use the same port number. These collisions could cause problems if the channel is run over legitimate traffic [10].

Bandwidth for this channel might appear to be limited because of the tendency of most transport layer protocols, particularly TCP, to use one socket per session. However, it is not atypical for multiple sessions to be generated per task. For example, when a typical web browser retrieves the HTML document, style sheet, images and other elements from a single web page it will frequently establish several sessions with the remote server so that it can make many requests at a time, enhancing protocol performance. Each of those requests uses a different ephemeral source port, meaning that simply accessing a lone web site with many elements can provide adequate cover for this channel at high bandwidths. Similarly, any protocol that takes advantage of parallel network sessions could support high bandwidths with this channel.

If the sender and receiver communicate on a regular basis the channel does not need to generate any new traffic. If they do not normally communicate, there is much flexibility in the traffic that can be used because of the application-neutral nature of the channel. Virtually any protocol can be selected for packet generation. This makes the channel simple to customize for any number of environments without raising the suspicions of common security appliances or analysts.

The channel does have a number of inherent weaknesses. For instance, the prolific use of network address translation technology (NAT) stands to limit the utility of the channel as described. This is because many NAT implementations modify the socket pair so that the source port received by the receiver cannot be reliably controlled. As such, if the sender lays behind a NAT box this channel is limited to communicating with other machines behind the NAT box. Similarly, proxy servers typically change the socket pair, again limiting the applicable scope of the channel. Sometimes the use of proxy servers is enforced even within a LAN, potentially crippling the channel. Legitimate traffic from the sender can possibly interfere with the channel in two separate ways. First, if another unrelated process makes use of an ephemeral port, that port will be locked from other processes until the TIME_WAIT timer expires. This timer, built into TCP with RFC793, is designed to ensure that the socket can still properly handle traffic arriving late from a closed connection [10]. If the source port required for the next data transmission is still in TIME_WAIT, a poorly written or light implementation might crash. While it is possible to work around this issue, higher level permissions are generally required. Second, if the sender and receiver communicate for some legitimate reason outside of the channel process it is possible that the receiver misinterprets the source port used in that exchange as part of the message, corrupting the data and calling to question the integrity of data received over this channel. This last problem can be effectively eliminated by using a robust encoding mechanism like the one discussed in subsection 4.2.

3.3 Potential Implementations

This method can be implemented in any number of ways ranging from the very clumsy to the very elegant. A simple implementation might generate false traffic with no real purpose other than providing a medium for the channel. Such an implementation would have high bandwidth but would be easy to identify as it would carry no changing data except for the source port. A more sophisticated version might act as a local wrapper for applications to use which would replace source port addresses for packets it receives and map it back to the original address, not unlike the basic functionality of NAT. Legitimate client applications could be a modified to take advantage of the channel. For example, a web browser can be modified to use the proper ephemeral port unless it is communicating with the intended receiving server, in which case it would use encoded delta ports instead of standard ephemeral ports.

A much more elegant approach than these might include a kernel level modification on the sender. For instance, every time any application communicates with the intended receiver, the sender kernel selects encoded delta ports. An implementation like this would eliminate the need to manage redundancy checking (discussed in section 4.2), greatly improving bandwidth while only using legitimate user traffic to transmit data.

4 Encoding Mechanisms

4.1 Simple Encoding Schemes

One example of a simple delta encoding scheme for this channel is to use the difference between two raw consecutive port numbers as the value to be transmitted. For example, if a user wanted to transmit the message "ABC" over the channel, they might first start a session with the source port 50,000, followed by 50,065, then 50,131, and finally 50,198. The differences between each port are 65, 66, and 67 respectively, which are the values of the ASCII decimal representations of the above message. This is represented visually in table 1 where the non-italicized bits carry the encoded data.

Table 1: Basic 8-bit Encoding of "ABC"

Port	Binary Representation
50,000	*1100 0011* 0101 0000
50,065	*1100 0011* 1001 0001
50,131	*1100 0011* 1101 0011
50,198	*1100 0100* 0001 0110

As mentioned above, IANA recommends that the first two bits be set to one for ephemeral ports and, although the range is not a technical limit, traffic coming from any port not adhering to this rule may trigger a signature in an intrusion detection system or fail to pass through an internal firewall [11]. For that reason this encoding scheme should comply with IANA recommendations, giving the scheme a port range between 49,152 and 65,535. Once the upper limit of this range has been reached, the numbers can loop around to the bottom range picking up where they left off. This function is described in Equation 1 where R_{min} and R_{max} are the range limits, V is the value to be transmitted, P_1 is the current port and P_2 is the next port to be used. For example, if the last port used was 65,500 (P_1) and the next value to be transmitted is 65 (V), it is clear that the port number is going to need to loop as the port 65,565 is beyond the upper limit. The difference of the 65,535 (R_{max}) and the last port used should be subtracted from the value to be transmitted. The sum of that difference and 49,152 (R_{min}) minus 1 is the next port to be used. In this case, the next port would be 49,181 (P_2).

$$P_2 = R_{min} + \left(V - (R_{max} - P_1) \right) - 1 \qquad (1)$$

This encoding scheme is somewhat inefficient. The problem is that no more than eight of the sixteen bits are ever being used at a time as the difference will never exceed 256. To increase efficiency while staying within the guidelines set forward by IANA, twelve or fourteen bits could be used on a rolling basis. Table 2 illustrates how a twelve bit implementation might encode the ASCII message "ABC". Note that the first four bits, shown in italics, are ignored. The remaining bits are concatenated with the other port bits and interpreted as a single binary string. A twelve bit encoding mechanism such as this would enjoy 50% better throughput that the eight bit counterpart outlined earlier, and a 14 fourteen bit representation would have 75% better throughput.

Table 2: Basic 12-bit Encoding of "ABC"

Port	Binary Representation
49,156	*1100* 0000 0000 0100
49,539	*1100* 0001 1000 0011
52,320	*1100* 1100 0110 0000

In some cases an implementation might not need to worry about the IANA port standard and would be free to use all sixteen bits. It may seem logical to simply divide the port bits in half and use the difference between them, but this method would forfeit the major benefit of delta encoding because the data would be completely contained in a single port number, making it easier for an analyst or security appliance to identify the channel and discover the data being transmitted. If a full sixteen bit scheme is selected, a better solution would be to use the delta between the first byte from two ports, followed by the delta between the first byte from the second port and the second byte of the first port. Finally, the delta between the second byte of the first port and the second byte of the second port is considered. At that point the pattern can be reversed and the cycle can continue. This is demonstrated in table 3.

Table 3: Simple 16-bit Encoding of "ABC"

Port	Binary Representation
131	0000 0000 1000 0011
19,654	0100 1100 1100 0110

4.2 Advanced Encoding Schemes

While functional, the above basic encoding methods can be problematic. The first major issue with these schemes, especially the eight bit scheme in particular, is that they are easy to identify. Second, they are all prone to data corruption. As discussed above, there is a risk that legitimate, unrelated communication between the sender and received could interfere with the channel by using source port numbers within the next delta range. There are 16,384 available ephemeral ports in the IANA suggested range, meaning the above eight bit implementation of the channel could be disrupted by an ephemeral selection of anywhere between 256 and 512 ports. This translates to a chance of data corruption between 1.56% and 3.13% for every unrelated source port number. While some practical implementations might be willing to call this acceptable loss in exchange for simplicity and bandwidth, there may be cases where a more robust approach is needed. In these cases improvements can be made to the encoding mechanism. One such improvement is defined below.

This more advanced encoding method uses the available bits left over from the data encoding scheme to help verify the contents of the next packet. In the previously discussed eight-bit scheme there remain eight bits in the sixteen bit port number which is further cut to six bits due to the IANA ephemeral port definition discussed above. The ones in the following bit string represent the bits in question: 0011 1111 0000 0000.

These bits will be used as a redundancy check (RC) to verify that the next source port received is, indeed, part of the message. To achieve this, an “exclusive or” (XOR) operation is run between the data bits of the current source port and the data bits of the previous source port. The resulting bit string is truncated to fit the available RC bits, depending on implementation. This method leaves the very first source port in the chain without data to XOR. To address this problem, both machines will share a key the same length as the RC bits. The RC bits in the first source port sent will be the result of an XOR between that key and the data bits to be transmitted. When these port numbers are considered in context, it is very easy to identify and ignore ports that are not a part of the message, greatly increasing data integrity. This process is illustrated in figure 1 and an 8 bit example is given in table 4. Note that the two leading IANA bits are in italics and ignored. The six bold bits for a given port are the result of an XOR operation between the data bits in that port and the data bits of the previous port or the initialization key.

Table 4: Advanced 8-bit Encoding of "ABC"

Port	Binary Representation
Key	10101010
49,643	*11* **000001** 11101011
50,478	*11* **000101** 00101110
57,201	*11* **011111** 01110001

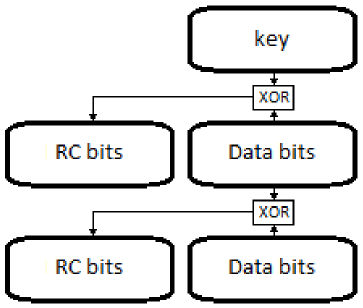

Figure 1: Advanced Encoding Process

This method allows only two bits worth of remaining offending ephemeral ports, or four ports total. This lowers the chance of data corruption to 0.02%. It should also be noted that this modified encoding scheme has the additional advantage of being more difficult for an analyst or security appliance to detect as it maintains the advantage of being present only in the delta while making the delta harder to discover and making the raw source ports jump around.

Once again, this encoding scheme stands functional but imperfect. If an implementation has no need to adhere to the IANA port standard, a much improved scheme can be developed. While maintaining eight data bits a full set of eight RC bits could be committed to the channel, leaving no chance of data corruption by unrelated traffic due to a perfect XOR. If the next legitimate port happens to be selected by an unrelated program, the real source port would be ignored due to incorrect RC bits, leaving no corruption. An example of this can be seen in table 5.

Table 5: Advanced 8-bit Encoding of "ABC"; not IANA Compliant

Port	Binary Representation
Key	10101010
16,875	**01000001** 11101011
50,478	**11000101** 00101110
24,433	**01011111** 01110001

Even if IANA standards must be adhered to, an improved implementation is possible with an encoding implementation which uses seven data bits instead of eight. There would be seven RC bits remaining to ensure integrity, leaving no chance of data corruption by unrelated traffic for the same reason outlined above. An example of this can be seen in table 6.

Table 6: Advanced 7-bit Encoding of "ABC"

Port	Binary Representation
Key	10101010
61,429	*11* **1011111** 1110101
57,163	*11* **0111110** 1001011
49,870	*11* **0000101** 1001110
61,200	*11* **1011110** 0010000

5 Conclusion and Future Work

This work presented a new method for leveraging transport layer source ports as a covert channel. A number of implementation models were discussed, including an efficient and covert kernel modification. Additionally, a wide variety of encoding schemes were proposed and reviewed on their merit. These contributions open the door to a number of new methods that warrant further work.

One method that may be worth exploring for future work is a channel which duplicates regular network traffic while changing the source port in the duplicate packet. Instead of using live packets as the medium for the channel, a local listener on the sender could wait for outgoing traffic destined for the receiver. Once the traffic is identified, the sender will create duplicate packets of the legitimate traffic, changing the source port in each packet to encode the leaking data. In this instance, the encoded delta is between the legitimate packet and the modified packet, as opposed to the delta between two modified packets. This approach has some advantages. First, legitimate traffic will not have any effect on the channel; whatever ephemeral source port is selected, the modified duplicate packet will be able to use whatever port it needs for the encoding as it will not actually open the socket advertised locally. Similarly, there is no need to worry about TIME_WAIT status of the sockets because the socket is never actually opened. Finally, this approach will allow a much higher bandwidth in a TCP environment as it will not need to establish a connection for each delta. The primary disadvantage to this technique is that it dissolves the features that make source port delta channels appealing from the perspective of covertness. There would be a high amount of unusual traffic over the network, making it easy to tell that some sort of communication is going on. Further, the new packets are exact duplicated of legitimate traffic except for the source port, making it easy for an analyst to identify the source ports as suspicious and possibly leading to the discovery of the transmitted data.

Another promising method involves using destination ports in UDP as a way to transmit data. On many UDP protocols, when a server receives a connection from a client it replies back with a new port listed for this particular client to use. This method allows UDP protocols to keep track of different “connections” without the benefit of TCP connectivity facilities. However, there is no limit to this port switching technique and it may be feasible to leverage rapid port switching deltas as a covert channel. There would be some distinct advantages to this method, including that the channel would survive NAT and proxy interference. Further, since the role of the sender and receiver is swapped, this method shows promise as a medium for covert command and control. A disadvantage associated with this method is that it would be inherently lossy.

Detection of this channel has yet to be researched. One approach could be comparing the rapidly changing port numbers to ordinary network traffic patterns. It may be possible to identify or prevent this channel by noting source port selection outside a standard variance.

6 References

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

[2] Johnson, D., Lutz, P. and Yuan, B., “Behavior-based covert channel in Cyberspace,” In: Vanhoof, K., et al (eds) *Intelligent Decision Making Systems*. World Scientific, New Jersey, pp. 311-318, 2009.

[3] S. Cabuk, C. E. Brodley and C. Shields, “IP covert timing channels: Design and detection”, in *Proceedings of the 11th ACM Conference on Computer and Communication Security*, Washington DC, USA, 2004.

[4] S. J. Murdoch and S. Lewis, “Embedding Covert Channels into TCP/IP”, in *Information Hiding Workshop Proceedings*, Berkeley CA, USA, 2005

[5] R. Bidou, F. Raynal, “Covert Channels,” (2009)

[6] J. Thyer, “Covert Channels Using IP Packet Headers”, presented at *DerbyCon*, 2011, Louisville, Kentucky.

[7] Stevens, W. R., B. Fenner, and A. M. Ruddof. *UNIX Network Programming: The sockets networking API*. Boston, MA: Pearson Education, 2004.

[8] Cotton, M., et. al. Internet Engineering Task Force , "Request for Comments: 6335 ." Last modified 2011. Accessed May 1, 2012. http://tools.ietf.org/html/rfc6335.

[9] Microsoft Corporation, "Important notice for users of Windows XP (SP3)." Last modified 2009 .http://support.microsoft.com/default.aspx?scid=kb;en-us;196271.

[10] Rfc 793: Transmission control protocol. (1981, September). Retrieved from http://tools.ietf.org/html/rfc793

[11] Firewall intrusion detection system signature enhancements. (n.d.). Retrieved from http://www.cisco.com/en/US/docs/ios/12_2t/12_2t15/feature/guide/ft_fwIDS.pdf