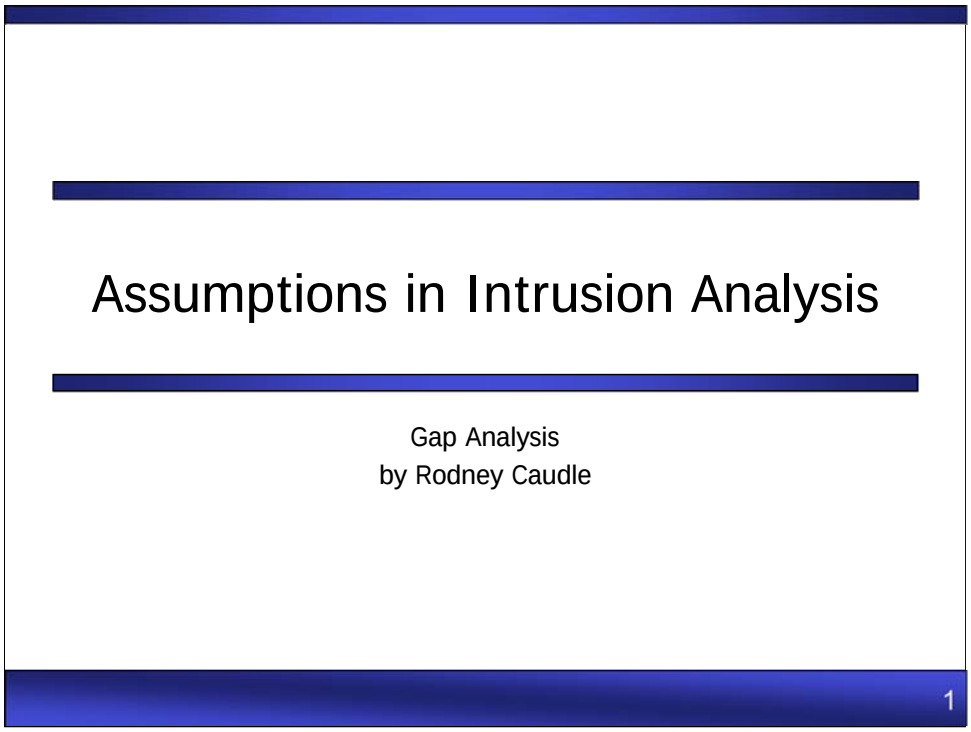
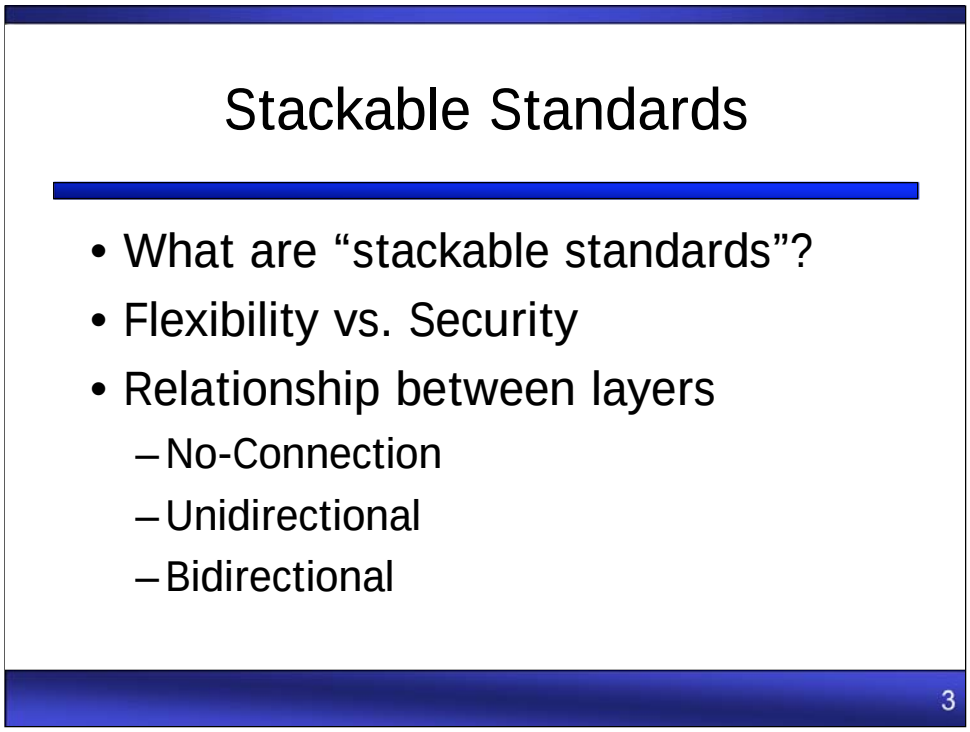


This presentation is based upon the research completed for the GIAC GCIA Gold paper for SANS. This paper received "honors" and provides a more in depth look at the information covered in this presentation. The link is provided below.

http://www.sans.org/reading_room/whitepapers/honors/1751.php?portal=862d604043cf137abf9e3d78795e25f4

Objectives

- What are stackable standards?
- Common Assumption in Analysis
- Discovering a Gap
- Exploiting the Gap

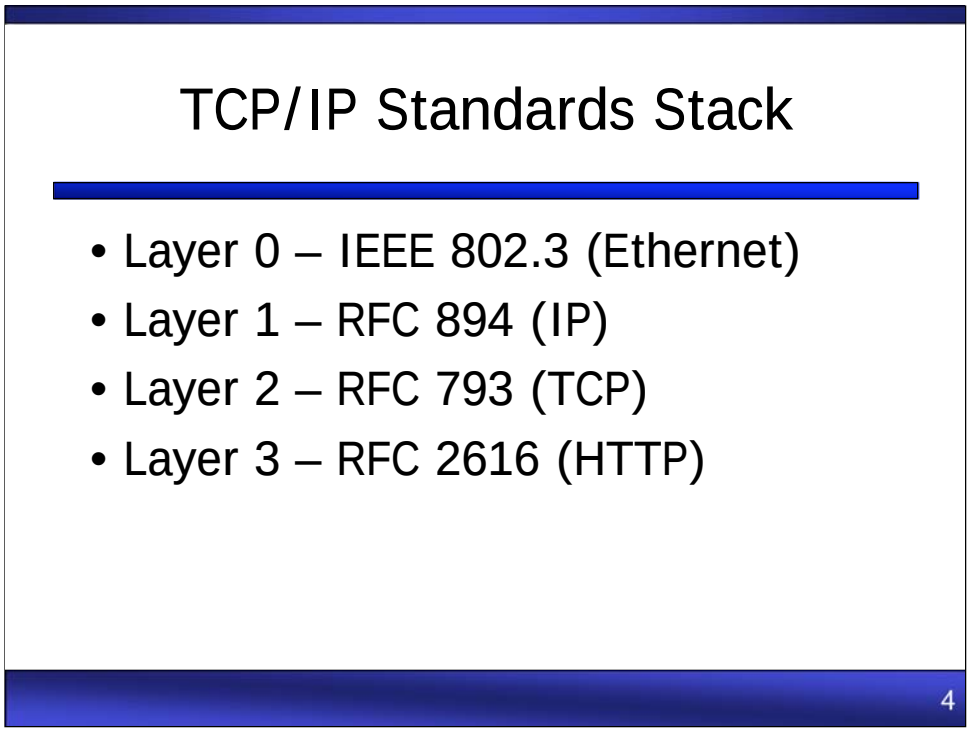


Stackable Standards are very common today. In fact, most communication that occurs today involves at least two different standards that are related some way into a “stack”. How the standards are related to each other is the focus of this paper. The stronger the rules defining how the layer inter-relate the less flexible the stack becomes. However, with strictly defined rules of interoperability the chances of exploiting the stack is lessened. There is always a tradeoff …

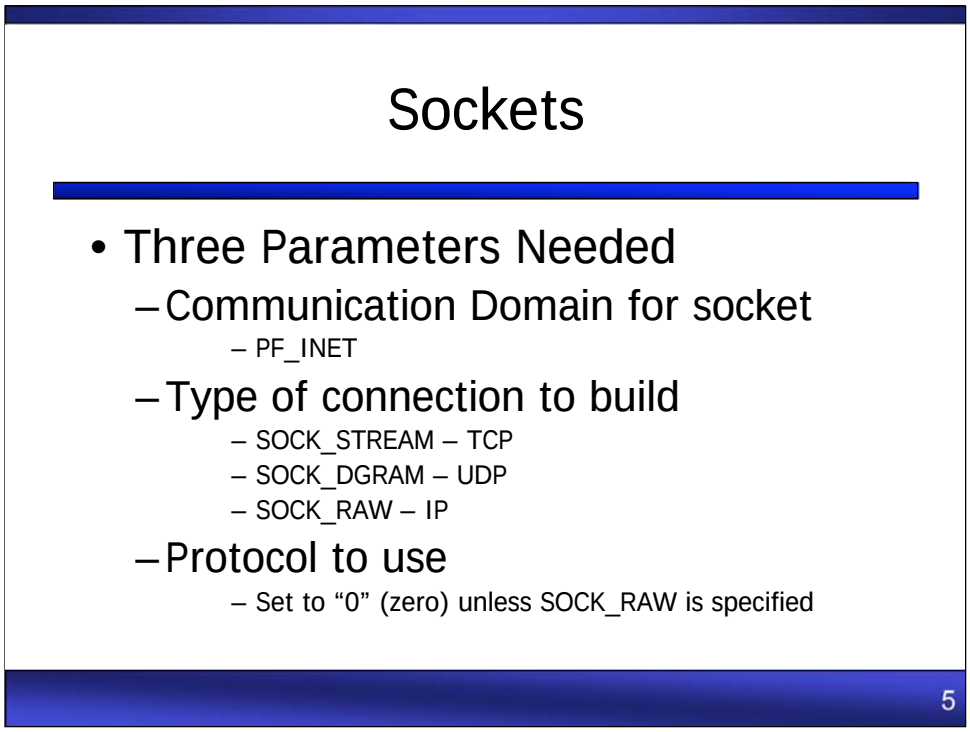
Consider Electronic Mail which involves two standards (disregarding the transport layers): Envelope and Message Body. There are no rules regarding the relationship of these two layers. Any type of message body can be sent with any type of header and the two standards do not rely on each other. This very flexibility is causing the increase in unsolicited bulk e-mail. An e-mail may be sent with an envelope specifying one address and a message body that states another. A loosely-coupled standard stack is why this is possible.

If Layer A is not tied to Layer B there will be no validation or dependence between the two layers. In this case the two layers are said to have “no connection” between each other. The two layers are essentially independent of each other and there is no cross referencing of fields between the two layers. An example of this type of protocol stack would be the standards used to facilitate the propagation of e-mail. The protocol stack used for e-mail consists of two layers, one for the “envelope” that contains the original message and one for the delivery of that envelope.

Another possible scenario is if Layer A is loosely tied to Layer B but Layer B is not tied to Layer A. In this case the rules of interaction are only one-way and the two layers would have a “unidirectional” connection. An example of this type of protocol stack is TCP/IP, specifically between Layer 1 (Network) and Layer 2 (Transport). Layer 1 uses a single field to control how the data is interpreted at Layer 2 when it is received. However, there is little validation performed on this interaction until the data is processed by Layer 2 on the receiving end of the communication.



We will focus on these standards: Ethernet, IP, TCP and HTTP. There are various standards available that can be used in place of each of these. For the purposes of this presentation we will focus only on these.



To facilitate communications between entities using TCP/IP each program must use sockets for communication. Sockets are a powerful way to configure communication endpoints. The declaration of a new communication endpoint requires three parameters and returns a descriptor which can be used for communication later.

Depending on the domain and type of socket created the visibility inside the packet(s) and the rules that are applied to the socket are different. The domain used in this paper is PF_INET which is valid for IPv4. The PF_INET domain of sockets accepts three types of sockets: SOCK_STREAM, SOCK_DGRAM and SOCK_RAW [17]. If a SOCK_STREAM is defined only TCP packets will be intercepted. If a SOCK_DGRAM is defined only UDP packets will be intercepted. For access to all other packets a SOCK_RAW socket must be defined. If a SOCK_RAW socket is defined the filtering is determined by the protocol. Common options for the protocol field are IPPROTO_IP, IPPROTO_ICMP, IPPROTO_TCP or IPPROTO_UDP.

In the following example, a raw socket is defined with a filter for only TCP packets.

```
#include <sys/types.h>
#include <sys/socket.h>
int main() {
                int testsocket;
                // open a raw IP socket
                testsocket = socket(PF_INET, SOCK_RAW, IPPROTO_TCP);
                // close the TCP socket
                close(testsocket);
}
```

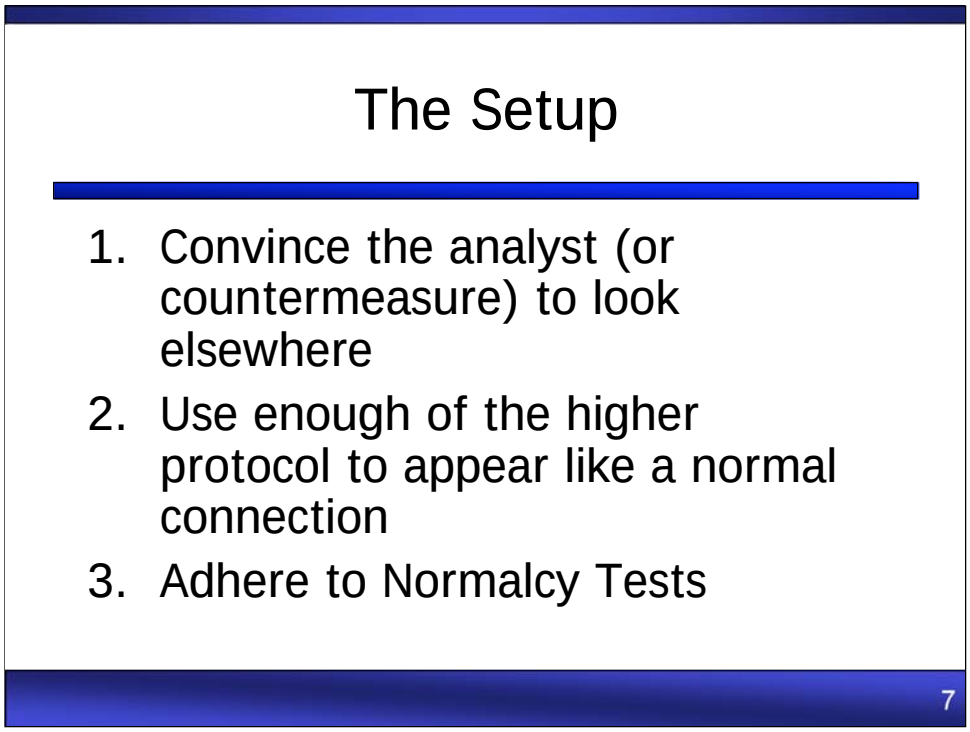
Assumptions

IP Protocol Field is trustworthy FALSE

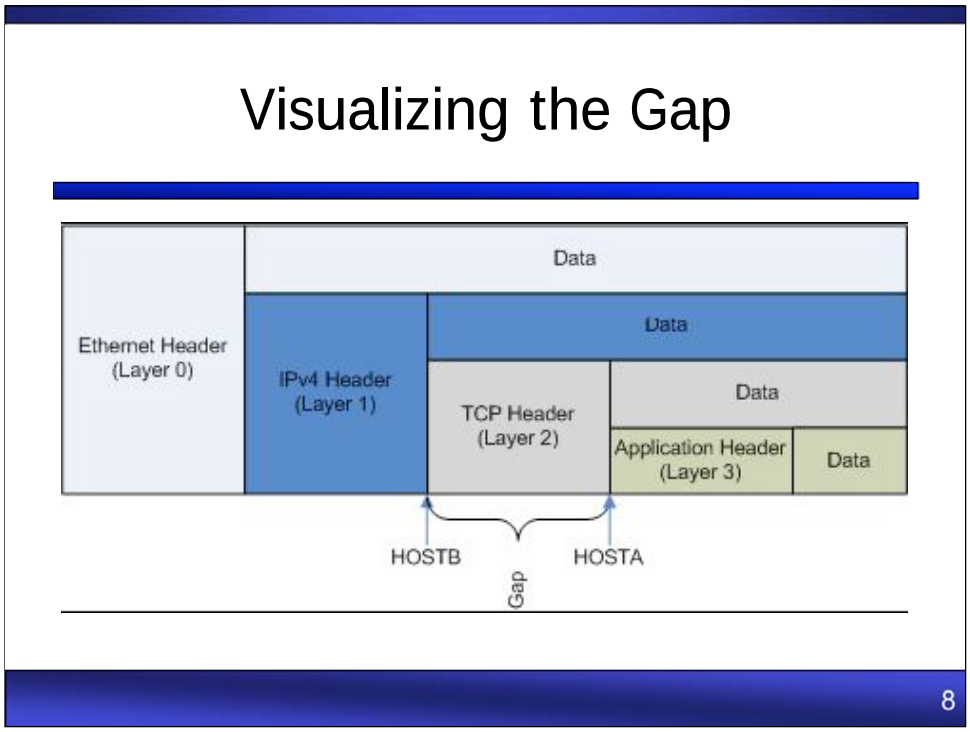
- There is no requirement for trusting this field beyond instructing the receiving stack how to interpret the Layer 2 protocol

6

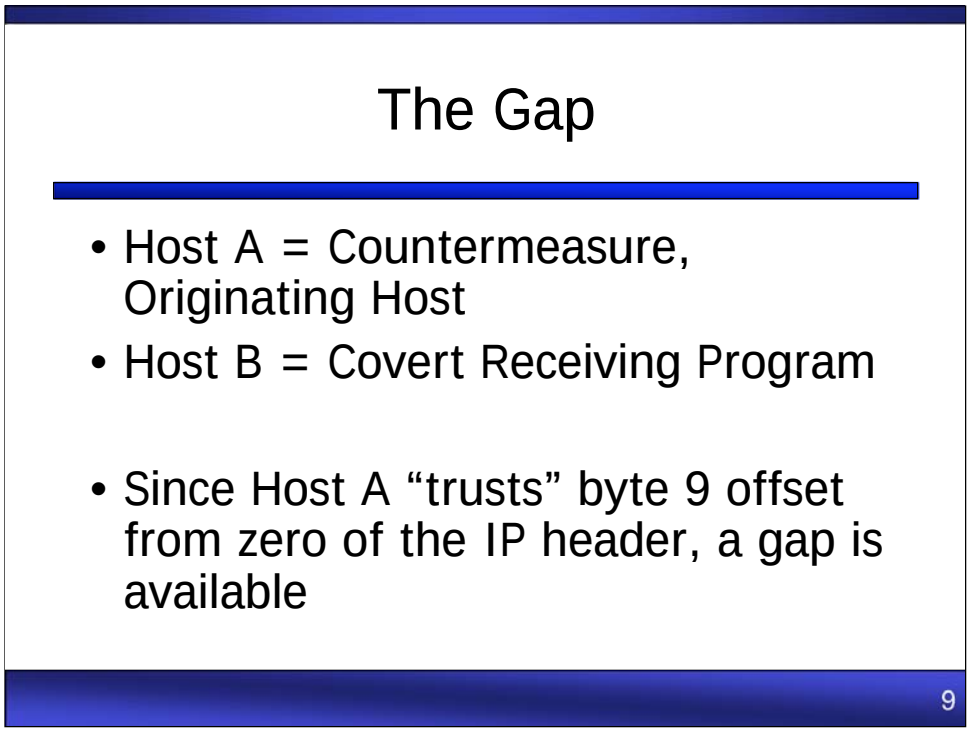
This assumption can be a misleading one. To understand why this is so consider the discussion earlier on the Sockets libraries for communication. Once the packets are dumped onto the network there is no follow-through required between hosts. This means that Host B can use a SOCK_STREAM socket and Host A can use a SOCK_RAW socket to communicate. This combination of sockets creates a gap in interpretation between the two hosts. Host B will be looking for data after the TCP header while Host A will begin looking for data after the IP header. With a properly crafted packet the IP Protocol field (byte 9 offset from 0 of the IP header) becomes the field utilized to mislead the countermeasures.



For this approach to be successful the gap provided by the assumption must be undetectable (ideal) or very hard to discern from normal communications. The first step to hiding this connection is to convince the analyst, or countermeasure, that the receiving host will connect at a higher point in the protocol stack. The second step is to use enough of the higher protocol header to appear like a normal connection attempt. Finally, an ideal connection will stay within a set of expected characteristics for normal communications. This includes ensuring that the checksums are accurate, ensuring the flags are setup, basically abiding by all the common “red flags” which are used by analysts and countermeasures to isolate rogue communications.



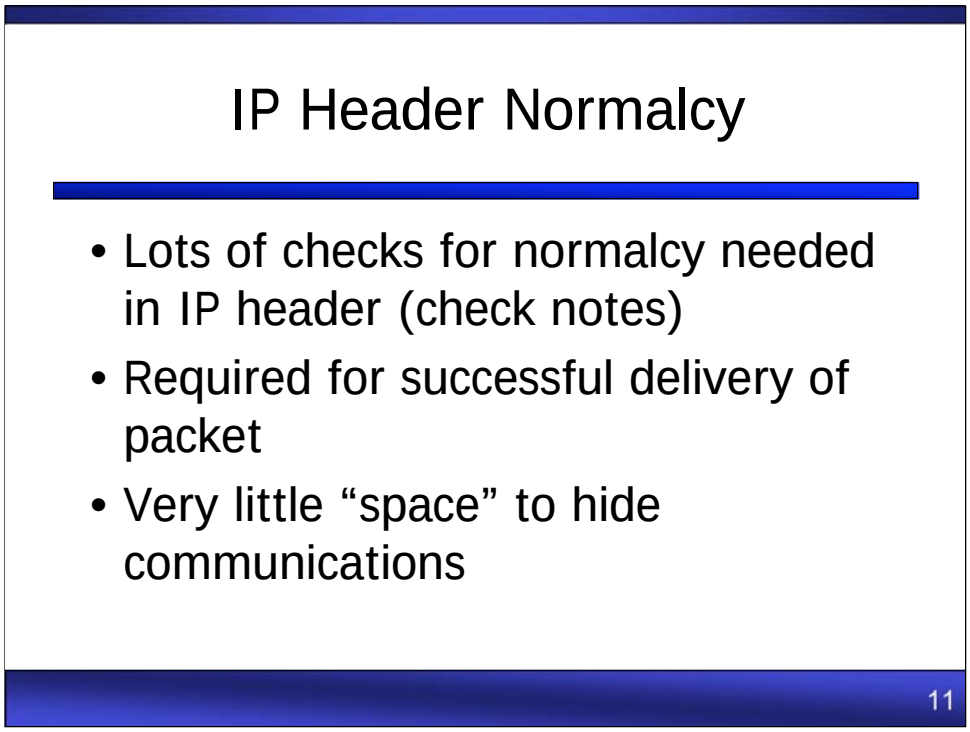
Host A could be a countermeasure (NIDS) or another host listening in on the conversation. If Host A looks through Layer 2 before considering "data" and Host B only looks through Layer 1 there is a Gap available for holding data. If this gap also adheres to the normalcy tests discussed earlier there is a good chance this data will never be found.



This gap provides the ability for the sending host to pass information to the receiving host and have it successfully pass through the network security layers. This is possible only if the network security countermeasures make an assumption about where the receiving host will connect in the protocol stack. In fact, this assumption is frequently made by security devices and security analysts, myself included (prior to conducting research for this paper of course). The assumption is directed by a single field in the IP header which defines how the upper layer protocol header should be interpreted. This field, byte 9 offset from zero, of the IP header, called the protocol field, is normally used to instruct the receiving host how to interpret the upper layer header. This field effectively provides direction for how to apply a bitmask across a predefined number of bytes before passing the data up the protocol stack. Based on the value of this field the size of the bitmask will vary. This paper takes a close look at this field and demonstrates why the assumption that this field is accurately set in connections is not a safe assumption to make.

Normalcy Tests

- What makes a packet appear normal?
  - Successful Delivery (functional)
  - No “red flags” which will direct analysts (or NIDS)



The version of normal communications should be set to depict IPv4 or IPv6 traffic. Any values outside of these two will be unusual and could attract unwanted attention.

The IP Header Length is a field which determines how long the IP header is. Unless IP Options are present this is set to five for a twenty byte header length. The presence of any IP options could attract unwanted attention.

The total length field should be accurate because inaccurate length fields could attract unwanted attention.

The flags should be set to reflect non-fragmented traffic since fragmented traffic is uncommon “in-the-wild” and can attract unwanted attention.

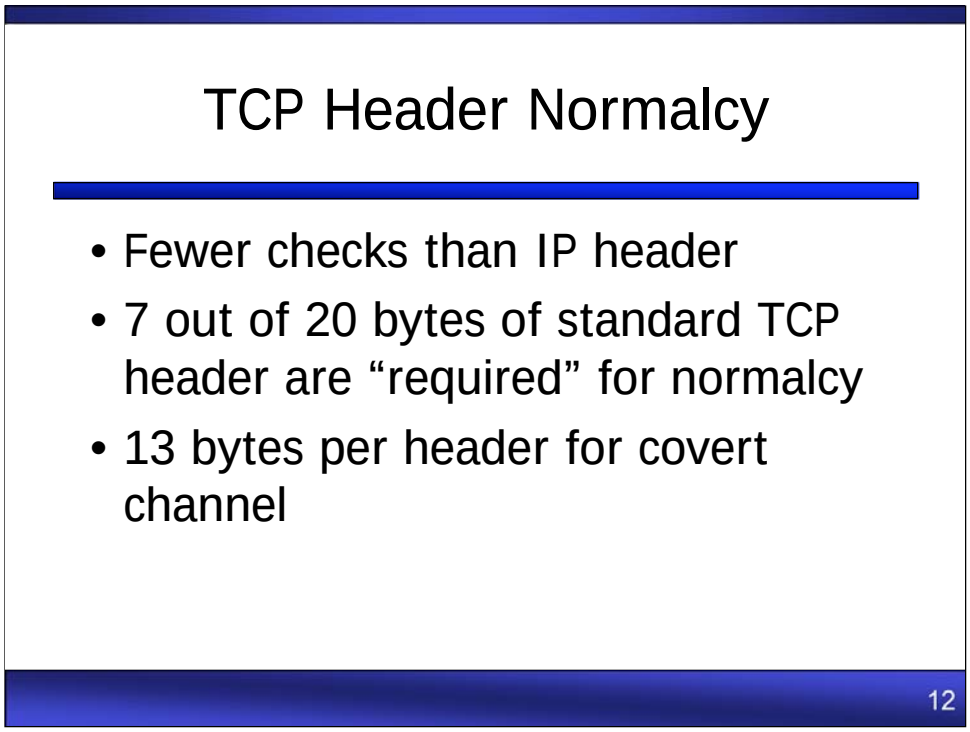
The TTL field should be set to the default for the OS the packet is being sent from. Setting the TTL to uncommon values could be an easy sign of crafting.

The protocol field should be set to the upper layer protocol in use.

The checksum should be accurate and functioning or the packet could be dropped before reaching the destination host by an intermediary host.

The source address should be the address of the sending host and should be a valid IP address for host which is initiating the connection attempt.

The destination address should be the address of the receiving host and should be a valid address for the network the packet is traversing.



The source port is needed to provide for identification of the connection attempt. While it is not a common check unusual source ports (low numbers) can attract unwanted attention.

The destination port is needed and should adhere to a common communication port. HTTP (80) is effective to use because the normally high volumes of traffic makes the connection harder to distinguish from the noise. Using unusual destination ports could result in a dropped connection and an easily identifiable signature.

The flags need to be preserved and should follow the TCP standard. Unusual combinations of TCP flags are normally associated with crafted packets.

The checksum should be valid. A bad checksum draws unwanted attention because it is not normal to find a packet with bad checksums unless errors are occurring on the network.

A SYN packet does not normally contain data. If data is found on a SYN packet the packet would draw unwanted attention.

Hiding the Channel

- Use the sequence number and acknowledgement number to embed data in the TCP header
- 12 bytes per packet to leak data
- Very hard to detect

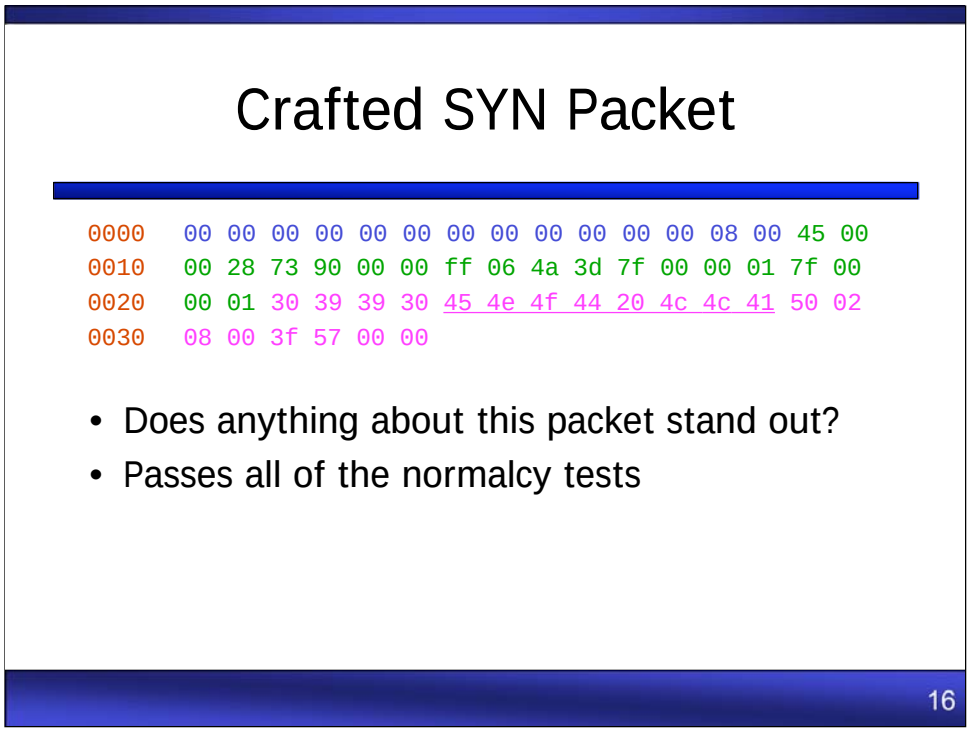
Getting Tricky

- What if only SYN packets are used?
  - Sequence and Ack will increment with each packet effectively making the channel disappear after the first packet

Normal SYN Packet

```
0000   00 00 00 00 00 00 00 00 00 00 00 00 08 00 45 00
0010   00 3c 42 5e 40 00 40 06 fa 5b 7f 00 00 01 7f 00
0020   00 01 83 0d 06 26 ad c2 f8 d6 00 00 00 00 a0 02
0030   80 18 a1 ea 00 00 02 04 40 0c 04 02 08 0a 03 4f
0040   ba 88 00 00 00 00 01 03 03 05
```

15



Results of Normalcy Tests:

==========

Version: Byte 0 of the IP header is set to “4” so this test passes.

IHL: Byte 1 offset from zero of the IP header is set to “5” for a 20 byte IP header so this test passes.

Total Length: Bytes 2 and 3 offset from zero is set to 0x0028 or 40 bytes so this test passes.

Flags: Byte 6 offset from zero is set to zero so there are no flags set so this test passes.

TTL: Byte 8 offset from zero is set to 0xFF (255) which is normal for a Linux host so this test passes.

Protocol: Byte 9 offset from zero is set to 0x06 which is TCP and is normal for a SYN packet so this test passes.

Header Checksum: The checksum is correct so this test passes.

Source Address: The source address is accurate so this test passes.

Destination Address: The destination address is accurate so this test passes.

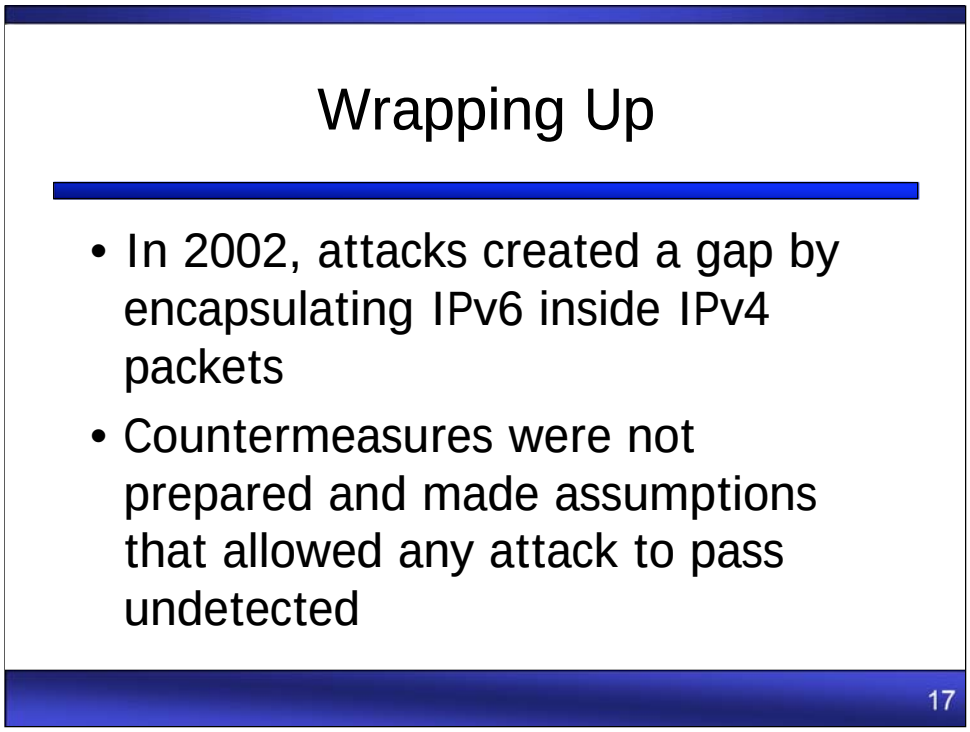
Source Port: Bytes 0 and 1 offset from zero of the TCP header is set to a valid ephemeral port so this test passes.

Destination Port: Bytes 2 and 3 offset from zero of the TCP header is set to an ephemeral port which is unusual and may post problems crossing a firewall. For more success the destination port should be set to a common port such as HTTP (80).

Flags: Byte 13 offset from zero of the TCP header contains the TCP flags and only the SYN flag is set so this test passes.

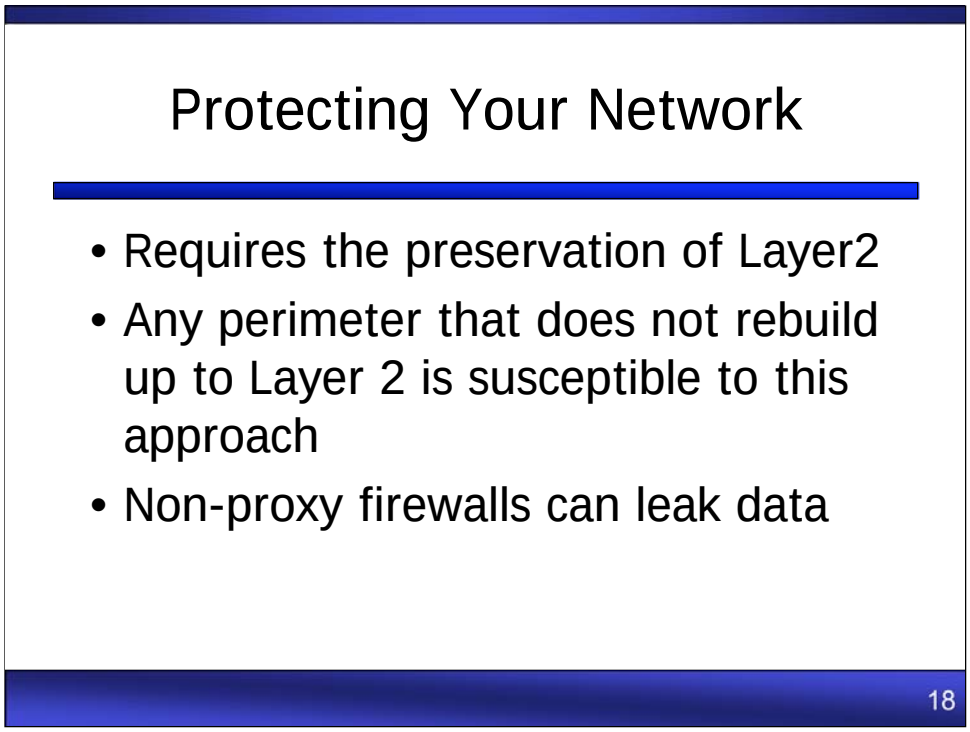
Checksum: The checksum is valid for this packet so this test passes.

Data On SYN: No data is present on this SYN packet so this test passes.



In 2002 there were many noted attacks which exploited a similar assumption in perimeter security devices. The description of attacks describes the encapsulation of an IPv6 packet tunneled inside an IPv4 packet. This effectively creates a gap because the countermeasures were developed to assume that an IPv4 header would be present and no signatures were built against an IPv6 header. However, the delivery devices made no such assumption and were able to handle either IPv4 or IPv6 packets meaning that, because the signatures did not match, the gap introduced by the IPv6 header enabled virtually any attack to pass undetected. The idea is to evade detection by developing a gap between where the initiating program begins interpretation and where the countermeasures or intermediary hosts begin interpretation.

The assumption discussed in this paper is that the information contained in the protocol field of the IP header is accurate and can be trusted. However, as this paper shows, this assumption is not always an accurate one. Due to the lack of a bidirectional connection between Layer 1 (IP) and Layer 2 (TCP) of the protocol stack the information cannot be trusted. This is not to say that the presence of a bidirectional connection is guaranteed to result in a trustworthy connection but the additional checks in a bidirectional connection make the information more trustworthy.



The approach discussed in this paper relies on the preservation of the upper layer protocol header. This reliance on the preservation of information for transmission provides characteristics which make this approach difficult to implement. First, any perimeter security countermeasure which rebuilds the connection at or above the upper layer protocol header will erase this covert channel in both directions. Second, due to the nature of the TCP protocol once a connection is established the portions of the header being used for the channel would need to be preserved to match the normalcy tests of a complete TCP connection. This approach would still work and could be used to embed a chat program inside a HTTP communication such that browsing various web pages would pass information on each SYN packet only.

The gaps introduced in loosely connected protocols are incredibly powerful tools for evading detection and will always be available unless future protocol stacks are built with bidirectional connections in mind. Until the stronger connections are available and multiple checks present in the protocol stacks it is recommended that a form of application-level countermeasure, such as a proxy-level firewall, be utilized as part of the perimeter security. The use of a proxy-level countermeasure will enforce the point of interpretation for packets entering and departing the perimeter. This would be the only effective way of preventing this approach from being successful.