A1 – Covert Socket Critiques And Improvements

Ben Soer

A00843110

Table of Contents

Craig Rowland's Covert TCP

Craig Rowland's "covert_tcp.c" contains a number of flaws in its design. Most notably Craig's horrible programming skills, causing the document to be painfully difficult to read through and decipher. Additionally he uses poor and inconsistent practices of using the standard libraries and raw socket libraries. Granted there is poor documentation in this area, Craig's code could be cleaned up significantly. A notable error in practice, is his use of creating a new raw socket for every character that is sent. By simply placing the socket call before the while loop, Craig could save considerable resources on the system. The Covert TCP code as well has a number of design flaws in its encoding of data into the TCP and IP headers. Firstly, Craig does not work whatsoever in making the Sequence Numbers or IP identification numbers look anything like their appropriate value. The header values are just set to the ASCII value of the letter being transferred. Decoding does the same procedure in pulling out the values. By implementing some kind of cipher, Craig could disguise the Ip ID much better. Additionally, implementing proper instrumentation of the Sequence numbers would greatly improve the covertness of sending with the SEQ and ACK headers. Second, Craig's implementation of the bounce server has one fatal flaw which he mentions in his paper. Whenever the SYN/ACK is sent to the destination server from the bounce server, a RST is sent back. This makes a distinguishable problem on the bounce server. If the sender does not own the bounce server, then the owners could quickly pickup a problem occurring on the network. A workaround this would be to use a UDP based protocol thus not requiring any state information to be passed. Another critique is that Craig's client and server are locked to each other. The client and server must both know the IPs of each other and the possible bounce server in between. Craig uses this information so as to be able to properly decipher the receiving and sent packets, but this greatly limits the portability of the code and setup time. There may be covert situations where a user either receiving or sending is unable to determine the source of the packets, but will know they are coming. A workaround would be to add a special encoding along with each packet to uniquely identify them among the traffic.

Covert Socket Improvements

For this assignment, Craig's program was modified to allow transfers of data via numerous different headers. Each header changed is presented in the following sections. Craig's basic framework in creating a packet and sending a single character from a file and then receiving that character and writing it to a file was kept for the transformation. Each header that was added to was then evaluated below on its positives and negatives in terms of covertly transferring data, or exposing the end system as suspicious. In addition, Craig's original implementation using the IP ID header and the TCP Sequence numbers are not evaluated as they were already previously done so in Craig's paper accompanied with the source code of this assignment.

Source Port

The source port is a very effective location of disguising the intended data. Source ports in ever new connection change and with enough ongoing connections can be quite fanatic in number choices. With 16 bits available the end systems can send 2 characters at a time. More covert options would be available in sending only 1 character at a time, allowing the client to be able to ensure the source port numbers are realistic and are always in the unprivileged port range. The source port can also be used in a bounce server, thus disguising the source of the packet from the recipient and recipient network.

A setback of the source port configuration is in the case of duplicates. Duplicate source ports from the same IP within a short time frame is quite rare and will likely get picked up on by any intrusion systems. A Work around for this would be to transfer only a single character and thus giving the client more options to ensure different source ports. Additionally spreading out the transfer so that repeats can appear more likely could mitigate the chances of being detected.

Destination Port

The destination port is not as effective as the source port. Firstly, it requires the server software to then be listening on all ports for data, making it harder to determine what is covert packets. Secondly, embedding ASCII characters will cause the destination ports to change, something that does not typically happen during any TCP session. Improvements could be made to disguise the packets to look like a TCP SYN DOS attack or a port scanner such as nmap, but this would quickly arouse suspicion on the recipient on why they are being attacked. Like the source port, the destination port can also be used in a bounce server, and thus disguise the source of the packet from the recipient.

Window Size

The window size offers a moderately effective covert location as it is not a regularly checked header unless there are problems already with the connection. With enough traffic in the network, these covert messages would likely go unseen for some time.

A setback of this design though is when sending information with largely different characters. This would cause dramatic changes in the interpreted window size and could trigger an intrusion detection system. The window size thus would be best suited for data that is very similar when in binary form. A transfer using morse code for example may be an ideal method of avoiding this. Additionally, as the window size is evaluated and set on each end system, it cannot be used in a bounce server architecture.

Flags

The flags pattern is highly effective in hiding the intended message, as it gives multiple options to scramble the bits that may make up the 8 bit message. However it is extremely noisy on the network as to get 8 bits requires the use of all 8 flags (SYN, FIN, URG, ACK, RST, PSH, RES1, RES2). This creates very obscure packets that would attract much attention from system admins and intrusion detection systems. On top of this, if the scrambling pattern is hard coded into the program, the program can effectively be fingerprinted by the network traffic and eventually traced, thus exposing the encoding pattern to the public.

To use the flags effectively, a subset of the flags could be used or a more sophisticated message sending pattern would be needed. The end systems could send using again a morse code like pattern and thus would only need to set 1 flag. Other patterns could be made by using more flags or creating more typical patterns. Alternatively, each end system could mimic actual data transfers, but doing so in a specific order that actually sends a message through the flags. The flag system could also be used on a bounce server system in some cases. A system will typically respond to a SYN with a SYN/ACK or a RST and so forth, thus by sending certain requests to the bounce server, the relayed message to the target could be parsed out of the appropriate reaction flags.

Ack Number

The ACK Number, like the sequence number is a very effective method of covertly sending messages. The ACK field allows for up to 32 bits or 4 characters to be sent at a time. By sending less characters, the messages could be more covertly sent by making the ACK numbers increment in more logical manners and represent more realistic ACK number values. Ideally the implementer should make ACK numbers that make sense with the transferred data, thus giving the least amount of suspicion and minimizing intrusion detection.

Time To Live

The TTL header, like the Window Size header is not often looked at, but would most likely be detected if its value radically changes between values. Most TCP transactions maintain the same TTL value. The TTL value is most commonly used to fingerprint operating systems. Having a transaction that uses obscure and radically changing values will likely catch intrusion detection and system administrator attention.

To best use TTL, use data that is the same or rarely changes. Morse code would be an ideal use for this situation. This would allow you to still create typically TTL values but transfer covert information still.

Type Of Service

The Type Of Service is a potentially ideal location for storing covert data but is highly vulnerable to failure as many routers and firewalls will override these values. Additionally, intrusion detention systems will either pick this up or reset these flags themselves, thus making your covert channel useless. Data in these flags is very rare and would like raise suspicion among any network administrator.