SimpleSPA

Applications Contents:

* VPNFinal.jar (this is the jar of the server side of the application)
* SPASimpleCSCompanion.jar (this is the jar of the client side of the app for Linux)
* SPASimpleCSCompanion\_windows.jar (this is the jar of the client side of the app for Windows)
* sampleSPA.conf
* source code files

SimpleSPA Application Overview:

This application consists of a single packet authorization mechanism designed for the purpose of hiding semi-public services like a SSH server. There is a server side (Linux only) and a client side (Windows and Linux). This app is similar to FWKnop and more of an academic/proof of concept app as opposed to full blown commercial quality app. It has however been tested extensively and I use it regularly as I travel frequently for work. I use it in conjunction with my SSH server. I keep all ports closed on the iptables firewall and allow SimpleSPA to briefly open the SSH port to allow for a connection and then close the port to new connections.

The app involves a client that creates a packet with a payload encrypted with the public half of two different RSA keys. The idea is that one key would be shared by all users and it would encrypt the user name of the individual. A second key specific to each individual user would encrypt a pre-shared key (just any old string, nothing secret about it really) and a timestamp (to counter replay attacks). The server would receive this packet and decrypt this first half of the packet…which would give us the user name of the person sending the packet. The server would then know which user specific second key to use to decrypt the pre-shared key and time stamp to evaluate them for acceptability. If all is good, then the server would open up a port for the semi-public service we were trying to conceal for a brief amount of time to allow for a connection to be made.

Upon receiving a successful packet, an iptables rule will be inserted allowing new connections for the port specified in the conf file. This assumes (probably should be parameterized) that iptables is located at /sbin/iptables. The type of rule inserted (again should also be parameterized, maybe next release) is like the following:

$IPTABLES –D –i eth0 –s <sourceIP> -p tcp –dport <port> -m state –state NEW,ESTABLISHED,RELATED -j ACCEPT

Another assumption I make is that your iptables firewall will have some kind of rule to allow established traffic to pass. The idea is to let SimpleSPA briefly open a port, let the user make the connection and then close the port, at least for new connections. In my firewall, I have rules like the following to allow for established traffic to pass.

#Allow established connections:

$IPTABLES -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT

$IPTABLES -A OUTPUT -m state --state ESTABLISHED,RELATED -j ACCEPT

$IPTABLES -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT

Step 4: Not included in this release.

Steps for Setup and Usage:

On the server, you will need to be running iptables at /sbin….not sure if any other Linux variants run it in other places, but BT4 runs it there so its good enough for me. You will also need to install the fun Sun JDK6. The repository version is fine and the apt-get command is shown below. You will also need to install a .de file the folks who wrote Jpcap (http://netresearch.ics.uci.edu/kfujii/Jpcap/doc/download.html), which is a Java wrapper/API for capturing packets. Yes the listener component of this app could easily have been written in C using Libpcap libraries. I did in fact do it this way initially, but quickly found that I am a better Java developer than I am a C developer.

Additionally you will need a jar from the the Bouncy Castle folks (<http://www.bouncycastle.org/download/bcprov-jdk16-145.jar>) who provide crypto libraries for creating the RSA keys for encrypting the payload of the single auth packet. The link for version 145 is posted above. I’ve been using version 138, cause I developed it a while ago. I will test sometime soon to make sure there are adverse consequences to using the latest version.

Server side config:

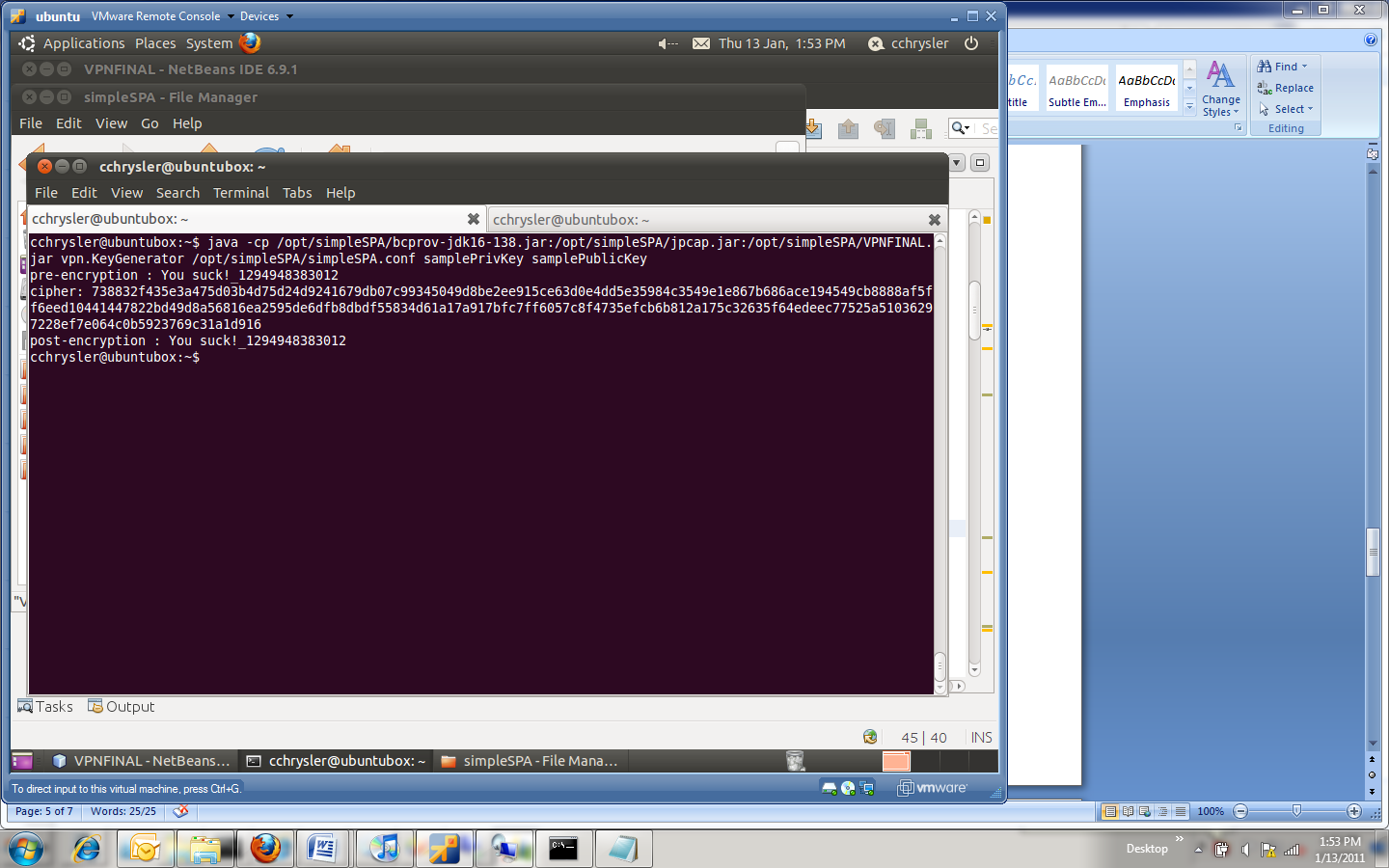
1. apt-get install sun-java6-jdk
2. dpkg --install jpcap-0.7.deb

Server side usage:

1. Create the conf file with all the appropriate variables.
2. Create a pair of keys of keys that all users will use and then a pair of keys for each specific user. Remember that the payload of single auth packet will contain strings encrypted with the public half of each of the keys.
3. Distribute the public key halves to the whoever the client will be and place the private keys in a file specified in the conf file that will contain string entries pairing up the user names to the private key file location (example. Bob.smith=/opt/simpleSPA/users/bobPrivKey)

Linux key generator call sample:

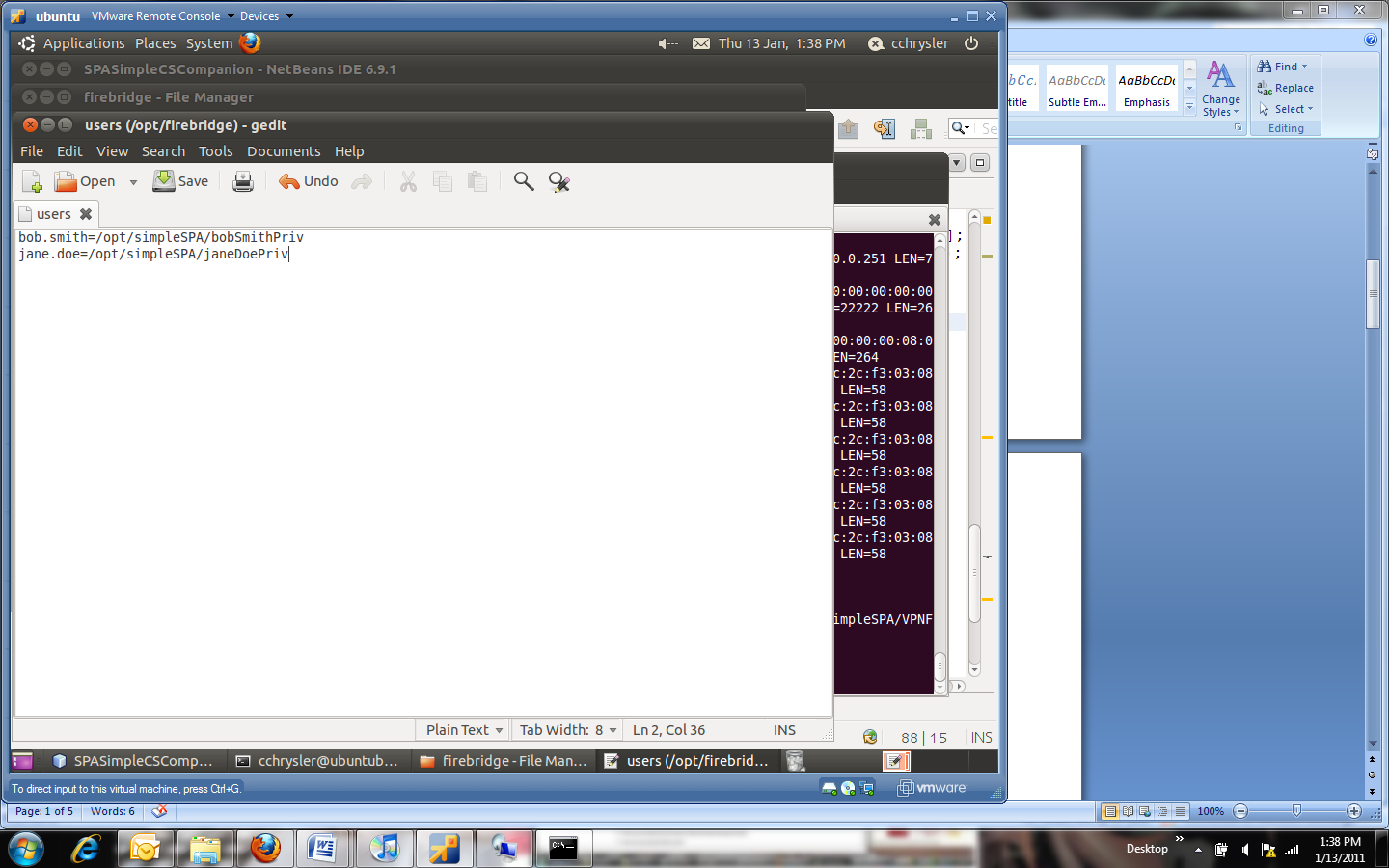
$ java –cp <location>bcprov-jdk16-138.jar:<location>VPNFINAL.jar vpn.KeyGenerator <conf file location> <private key name> <public key name>

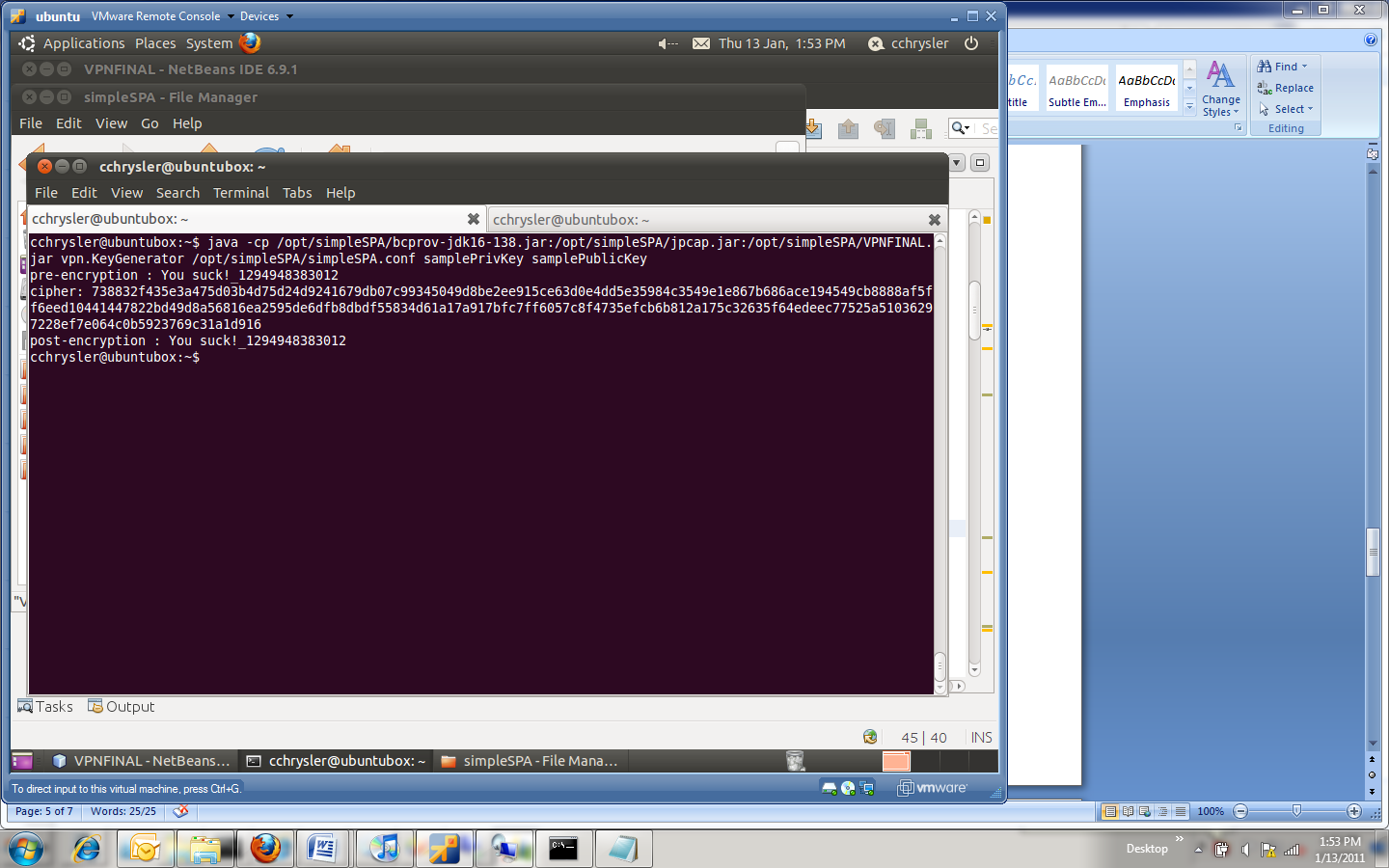


1. Setup the conf file.
2. Fire up the server. Write a script that will check if the process is running and restart it if by chance happened to crash….or else you will locked out of your SSH, or whatever service you are hiding ☺

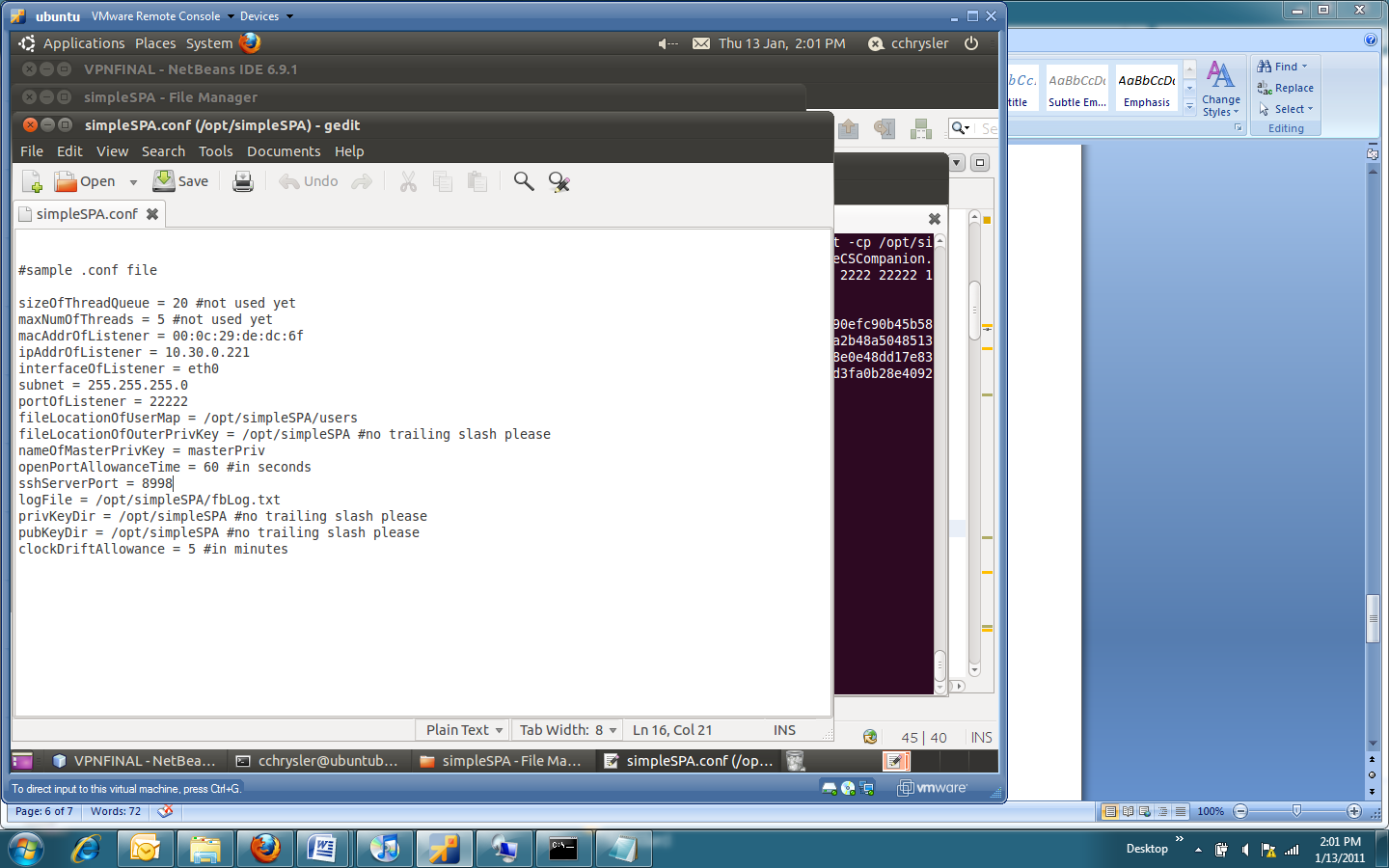
Linux SimpleSPA call sample:

$ java –cp <location>bcprov-jdk16-138.jar:<location>VPNFINAL.jar vpn.KeyGenerator <conf file location> <private key name> <public key name>





Sample Conf file



Windows Client Side Usage

The Windows side is very straight forward.

Windows client side call sample:

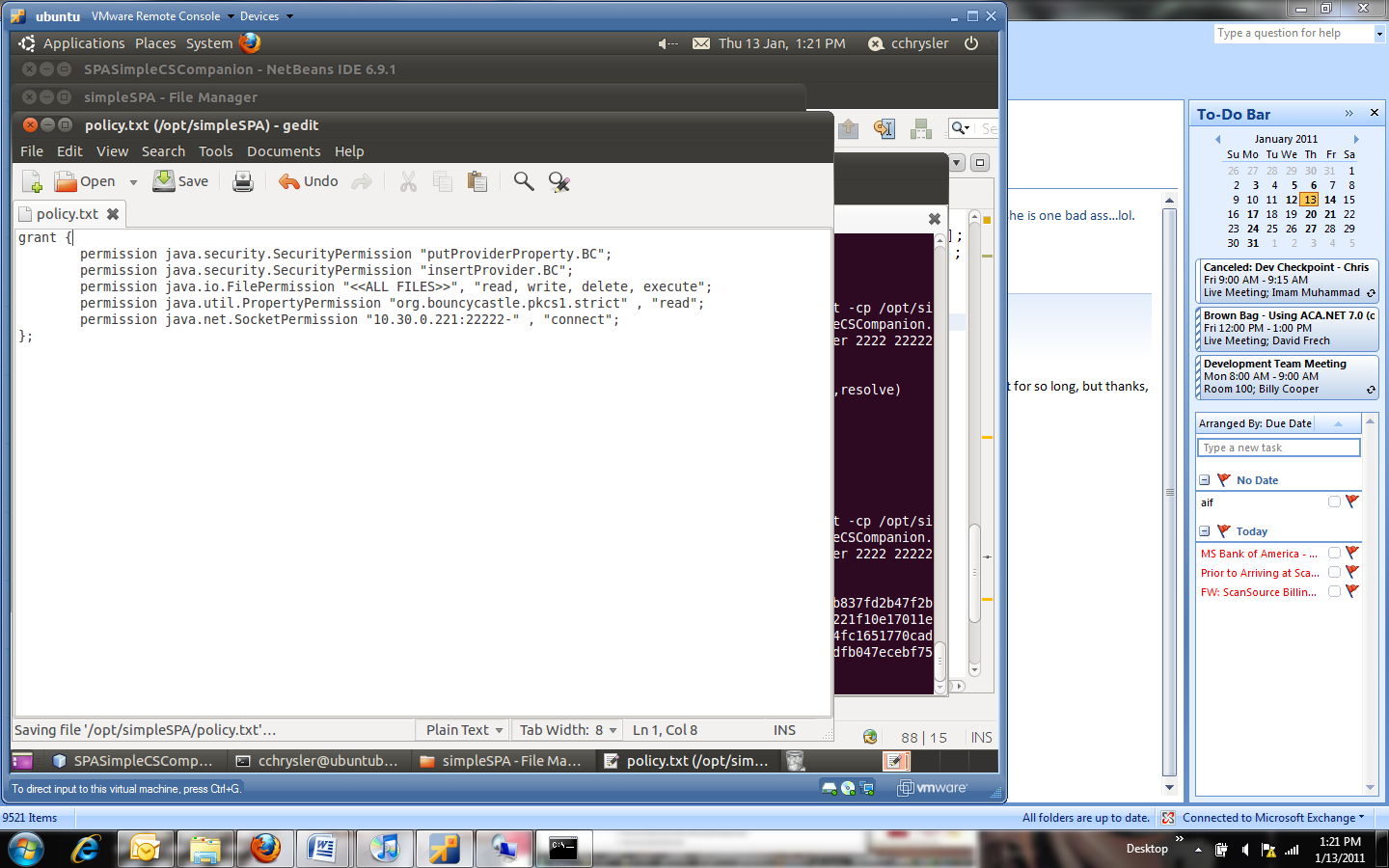
c:\>java –cp <location>\bcprov-jdk16-138.jar;<location>\SPASimpleCSCompanion\_windows.jar spa.UDPPacketAuthStandAlone\_v2 <source IP> <master public key> <user specific public key> <username> <source port> <destination port> <destination IP>

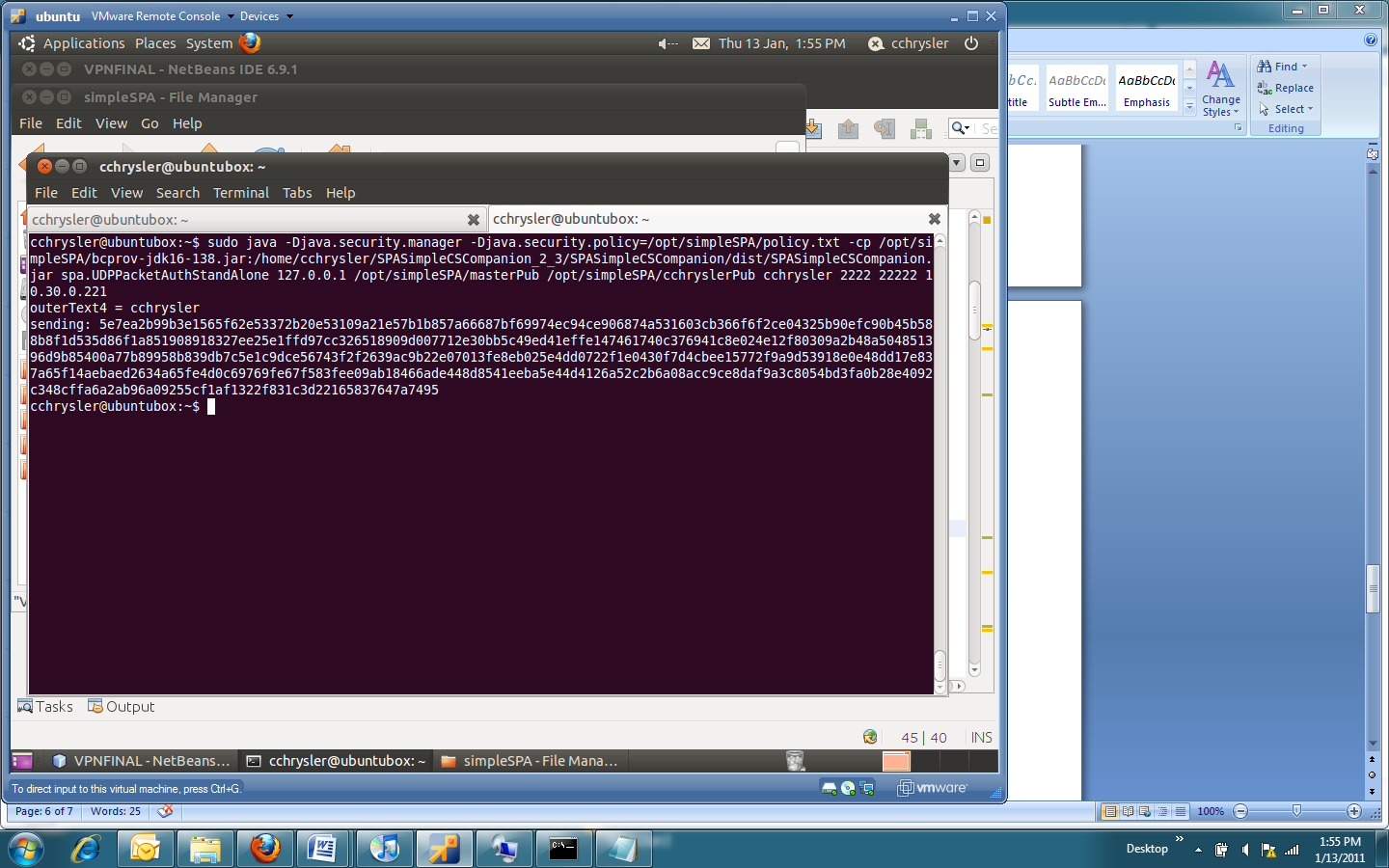
Linux Client Side Usage:

Linux is a bit more picky with the client side. You have to specify a policy explicating stating some permissions. Maybe there is a better way to do this, I don’t deal with permissions and policy files very often.

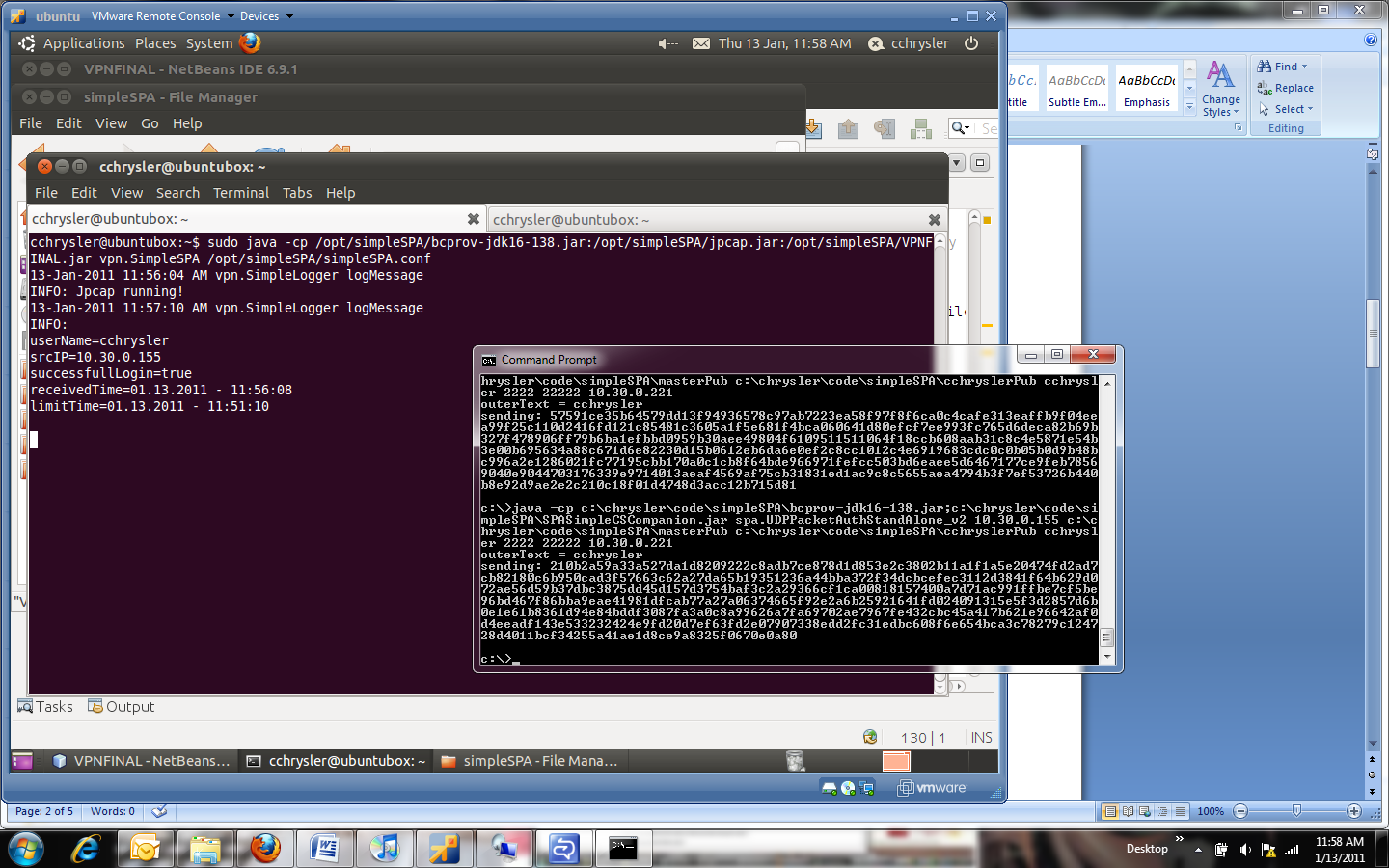
Linux client side call sample:

$ java –Djava.security.manager -Djava.security.policy=<location> –cp <location>bcprov-jdk16-138.jar:<location>SPASimpleCSCompanion.jar spa.UDPPacketAuthStandAlone <source IP> <master public key> <user specific public key> <username> <source port> <destination port> <destination IP>





The whole thing in action:



It is setup to log to a file in xml format.

