

Getting the Most Out of your NI Linux Real Time Target Tutorial



Contents

Securing you target with IPtables	2
Using Syslog for Event Logging	4
Using LAMP (Apache, PHP, MySQL) on NI Linux Real Time	6
File Encryption with GnuPG	9
Sharing files between cRIOs and Windows Machines	13
Code Reuse (C/C++)	18
Code Reuse (Python).....	20
Code Reuse (node.js)	24
Appendix A	27
Appendix B	28

Securing your target with IPtables

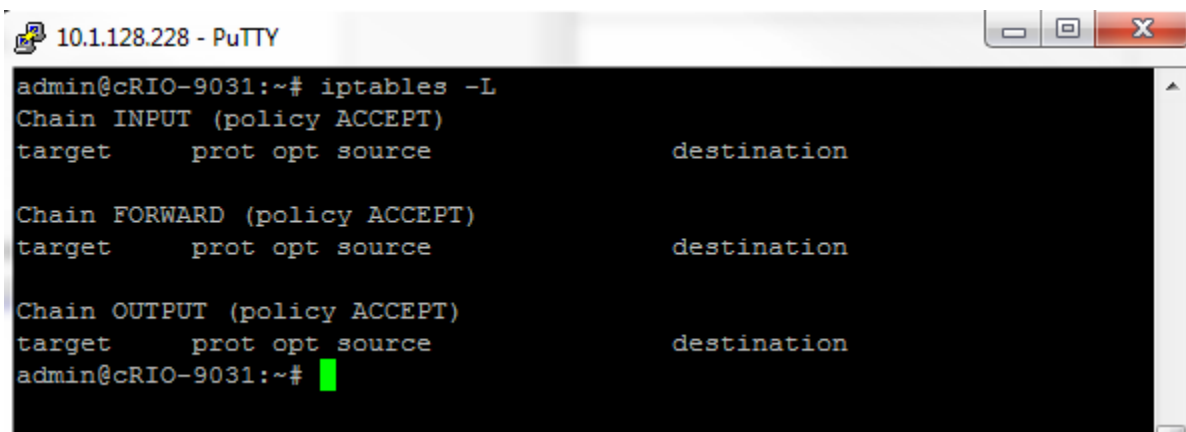
1. Installation

IPtables come pre-installed with NI Real Time Linux. Once the default software is installed through MAX you can start using iptables on your NI Real Time Linux target.

2. Getting Started

In order to start using iptables you need to access the console on the linux target through ssh or serial port and login with your NI-Auth credentials (default user: admin password: (blank))The following command lists current firewall rules for your target:

```
iptables -L
```



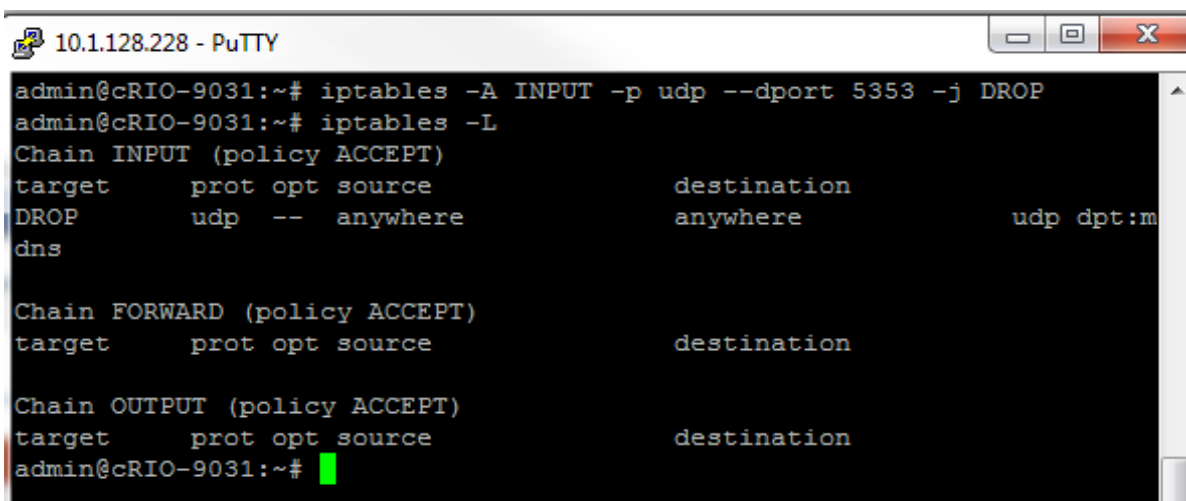
```
admin@cRIO-9031:~# iptables -L
Chain INPUT (policy ACCEPT)
target      prot opt source                destination

Chain FORWARD (policy ACCEPT)
target      prot opt source                destination

Chain OUTPUT (policy ACCEPT)
target      prot opt source                destination
admin@cRIO-9031:~#
```

For example to block port 5353 on a target console (serial or ssh), run:

```
iptables -A INPUT -p udp --dport 5353 -j DROP
```



```
admin@cRIO-9031:~# iptables -A INPUT -p udp --dport 5353 -j DROP
admin@cRIO-9031:~# iptables -L
Chain INPUT (policy ACCEPT)
target      prot opt source                destination
DROP        udp  --  anywhere              anywhere             udp dpt:m
dns

Chain FORWARD (policy ACCEPT)
target      prot opt source                destination

Chain OUTPUT (policy ACCEPT)
target      prot opt source                destination
admin@cRIO-9031:~#
```

In general you want to create a script that adds all the firewall rules to iptables. You can name the script firewall.sh and run it in the console:

```
1  #!/bin/bash
2  set -e
3  ## reset chain
4  iptables -F INPUT ACCEPT
5  iptables -F
6
7  ## Allow secure protocols
8  iptables -A INPUT -p tcp --dport 22 -j ACCEPT      # ssh
9  iptables -A INPUT -p tcp --dport 443 -j ACCEPT     # https
10
11 iptables-save > iptables.conf
```

After you run the script copy the generated file “iptables.conf” to
/etc/natinst/share/iptables.conf and reboot the target to apply the rules.

To enable the firewall:

Create /etc/natinst/share/iptables.conf, then reboot or
run /etc/init.d/firewall restart.

To temporarily disable the firewall:

On a target console (serial or ssh), run /etc/init.d/firewall stop.

To permanently disable the firewall:

Move or delete /etc/natinst/share/iptables.conf, then reboot or
run /etc/init.d/firewall restart.

References:

1. http://people.netfilter.org/hawk/presentations/devconf2014/iptables-ddos-mitigation_JesperBrouer.pdf
2. http://www.liquidcomm.net/news/tech_tips/linux_os/how-to-manage-a-ddos-or-dos-attempt-directed-at-your-linux-server.html
3. <http://serverfault.com/questions/410604/iptables-rules-to-counter-the-most-common-dos-attacks>

Using Syslog for Event Logging

1. Installation

Syslog-ng comes preinstalled with NI Linux Real Time.

2. Getting Started

To start using syslog you will need to create/modify `/etc/syslog-ng/syslog-ng.conf` (LabVIEW 2016 and earlier) or `/etc/syslog-ng.d/syslog-ng.conf` (LabVIEW 2017 and newer) for your specific logging configuration. The default `syslog-ng.conf` file located in `/etc/syslog-ng/` that comes pre-installed with NI Linux Real Time has a number of sources (ex. system kernel log `"/proc/kmsg"`) that syslog-ng collects messages from and stores the data in appropriate destinations (ex. `"/var/local/natinst/log/kern.log"`). Any messages that don't have a specific destination configured get logged to `"/var/log/messages"`.

To view the content of log files created by syslog you can view an appropriate file in a text editor or use NI Web Configuration and Monitoring interface by navigating to the IP address of your NI Linux Real Time target and clicking the "System Log Viewer" tab.

The screenshot displays the NI-cRIO-9034 System Log Viewer web interface. The main area shows a table of logged events with the following columns: Timestamp, Message, and Source. The table lists various system events, including cron jobs, user sessions, and deployment codes. On the right side, there is a 'Log Limit' section set to 10 and a 'Logs' section with checkboxes for different log sources: authlog, cron, messages, nimerlog, nikern, and nimerdms. The interface also includes a 'Refresh' button and a 'Restart' button in the top right corner.

Timestamp	Message	Source
7/25/2016 1:30:01 PM	crond[19420]: pam_unix(cron:session): session opened for user root by (uid=0)	authlog
7/25/2016 1:30:01 PM	CROND[19420]: pam_unix(cron:session): session closed for user root	authlog
7/25/2016 1:30:01 PM	CROND[19421]: (root) CMD (/usr/bin/logrotate /etc/logrotate.conf)	cron
7/25/2016 1:25:01 PM	crond[18998]: pam_unix(cron:session): session opened for user root by (uid=0)	authlog
7/25/2016 1:25:01 PM	CROND[18998]: pam_unix(cron:session): session closed for user root	authlog
7/25/2016 1:25:01 PM	CROND[18999]: (root) CMD (/usr/bin/logrotate /etc/logrotate.conf)	cron
7/25/2016 1:20:01 PM	crond[18533]: pam_unix(cron:session): session opened for user root by (uid=0)	authlog
7/25/2016 1:20:01 PM	CROND[18533]: pam_unix(cron:session): session closed for user root	authlog
7/25/2016 1:20:01 PM	CROND[18534]: (root) CMD (/usr/bin/logrotate /etc/logrotate.conf)	cron
7/25/2016 1:15:01 PM	crond[18109]: pam_unix(cron:session): session opened for user root by (uid=0)	authlog
7/25/2016 1:15:01 PM	CROND[18109]: pam_unix(cron:session): session closed for user root	authlog
7/25/2016 1:15:01 PM	CROND[18110]: (root) CMD (/usr/bin/logrotate /etc/logrotate.conf)	cron
7/25/2016 1:13:30 PM	sshd[17973]: pam_unix(sshd:session): Unable to look up user "admin"	authlog
7/25/2016 1:13:30 PM	sshd[17973]: pam_unix(sshd:session): session opened for user admin by (uid=0)	authlog
7/25/2016 1:10:01 PM	CROND[17678]: (root) CMD (/usr/bin/logrotate /etc/logrotate.conf)	cron
7/25/2016 1:07:36 PM	DONT DEPLOY CODE: "DevClass=cRIO" "ProdName=cRIO-9034" "SerialNo=01ACD241" "OS=NI-Linux x64" "ProdI	messages
7/25/2016 1:05:01 PM	CROND[17255]: (root) CMD (/usr/bin/logrotate /etc/logrotate.conf)	cron
7/25/2016 1:00:01 PM	CROND[16791]: (root) CMD (/usr/bin/logrotate /etc/logrotate.conf)	cron
7/25/2016 12:55:01 PM	CROND[16368]: (root) CMD (/usr/bin/logrotate /etc/logrotate.conf)	cron
7/25/2016 12:50:01 PM	CROND[15945]: (root) CMD (/usr/bin/logrotate /etc/logrotate.conf)	cron
7/25/2016 12:49:06 PM	DONT DEPLOY CODE: "MAC=00:80:2F:23:13:7B" ; ttl=4500]	messages
7/25/2016 12:48:38 PM	DONT DEPLOY CODE: "MAC=00:80:2F:23:13:7B" ; ttl=4500]	messages
7/25/2016 12:48:37 PM	DONT DEPLOY CODE: "DevClass=cRIO" "ProdName=cRIO-9034" "SerialNo=01ACD241" "OS=NI-Linux x64" "ProdI	messages
7/25/2016 12:48:31 PM	DONT DEPLOY CODE: "DevClass=cRIO" "ProdName=cRIO-9034" "SerialNo=01ACD241" "OS=NI-Linux x64" "ProdI	messages
7/25/2016 12:48:31 PM	DONT DEPLOY CODE: "DevClass=cRIO" "ProdName=cRIO-9034" "SerialNo=01ACD241" "OS=NI-Linux x64" "ProdI	messages
7/25/2016 12:48:29 PM	DONT DEPLOY CODE: "DevClass=cRIO" "ProdName=cRIO-9034" "SerialNo=01ACD241" "OS=NI-Linux x64" "ProdI	messages
7/25/2016 12:48:28 PM	DONT DEPLOY CODE: "MAC=00:80:2F:23:13:7B" ; ttl=4500]	messages
7/25/2016 12:48:18 PM	DONT DEPLOY CODE: "MAC=00:80:2F:23:13:7B" ; ttl=4500]	messages
7/25/2016 12:45:01 PM	CROND[15480]: (root) CMD (/usr/bin/logrotate /etc/logrotate.conf)	cron
7/25/2016 10:17:57 AM	kernel: [1.659666] Warning: unable to open an initial console.	nikern
7/22/2016 5:50:46 PM	kernel: [1.659543] Warning: unable to open an initial console.	nikern
7/22/2016 1:40:22 PM	kernel: [1.659658] Warning: unable to open an initial console.	nikern
7/22/2016 1:26:40 PM	n.1633.41 LabVIEW Real-Time process restarted.	nikern

Sending a message to syslog-ng through LabVIEW

To send a message to syslog-ng through LabVIEW you will need to complete the following steps:

1. Download the example source code from: <https://github.com/LabVIEW-DCAF/syslog>
2. Copy the appropriate [liblv_syslog_x86_64.so](#) (x86 Intel) or [liblv_syslog_armv7l.so](#) (Arm) file to /usr/local/lib on your NI Linux Real Time target
3. Open “syslogger.lvproj” from the example folder (step 1) and set the appropriate IP address for the Real Time target in the project
4. Run write syslog.vi

With the default configuration messages sent using the “write syslog.vi” will get logged to “/var/log/messages” destination.

References:

1. <https://syslog-ng.org/>

Using LAMP (Linux, Apache, MySQL, PHP) on NI Linux Real-Time

The bash script “LAMP_install_script_2018.sh” will install the LAMP stack on a NI Linux Real-Time target. The text of the script can be copied and pasted into the terminal, or script files (.sh) can be transferred onto the target and run from the terminal.

This script has been tested with the default 2018 LabVIEW Real-Time software stack and may need to be modified for use with other versions.

<https://forums.ni.com/t5/NI-Linux-Real-Time-Documents/Getting-the-Most-Out-of-your-NI-Linux-Real-Time-Target/ta-p/3523211>

1. Preparing the cRIO for LAMP stack installation.

The package `packagegroup-core-sdk-dev` includes the compiler, which will allow us to install applications from source code.

Python must be reinstalled from source, or the later installation of PHP in the script will not complete successfully.

2. MariaDB

MariaDB is a stand-in replacement for MySQL in the LAMP stack. It is available from the OPKG package manager.

As of June 2018, there appears to be a bug in the MariaDB installation version 5.5.55 from OPKG package manager and a helper script `mysqld_safe_helper` is missing from the installation. See this link for more details:

<https://patchwork.openembedded.org/patch/145834/>.

Workaround:

Starting the service with command `/etc/init.d/mysqld start` throws an error and fails to start the service. Use the command `mysqld` to start the service successfully. However, this puts the terminal in an unresponsive state and a new session must be opened to continue sending commands to terminal.

3. Apache

Apache installation requires Apache Portable Runtime Library, Apache Portable Runtime Library Additional Tools, and Perl.

Apache is configured by default to run on port 80, but we configure it to use port 8081 in this script to avoid conflict with NI Web-Based Configuration and Monitoring Service (WIF), which runs on port 80 by default. This is done by editing the Apache config file `/usr/local/apache2/conf/httpd.conf`.

Test successful installation of Apache by going to <http://<TARGET IP>:8081/index.html>, where <TARGET IP> is replaced with the IP address of your target, e.g. 10.1.128.6. Navigating to this URL should show the contents of the `index.html` file in the Apache `htdocs` folder, by default the text "It works!".

4. PHP

PHP installation requires XML C Parser.

Apache `httpd.conf` file needs to be edited to work with PHP.

The script creates a test script. If PHP install is successful, `http://<TARGET IP>:8081/test.php` will show PHP configuration settings.

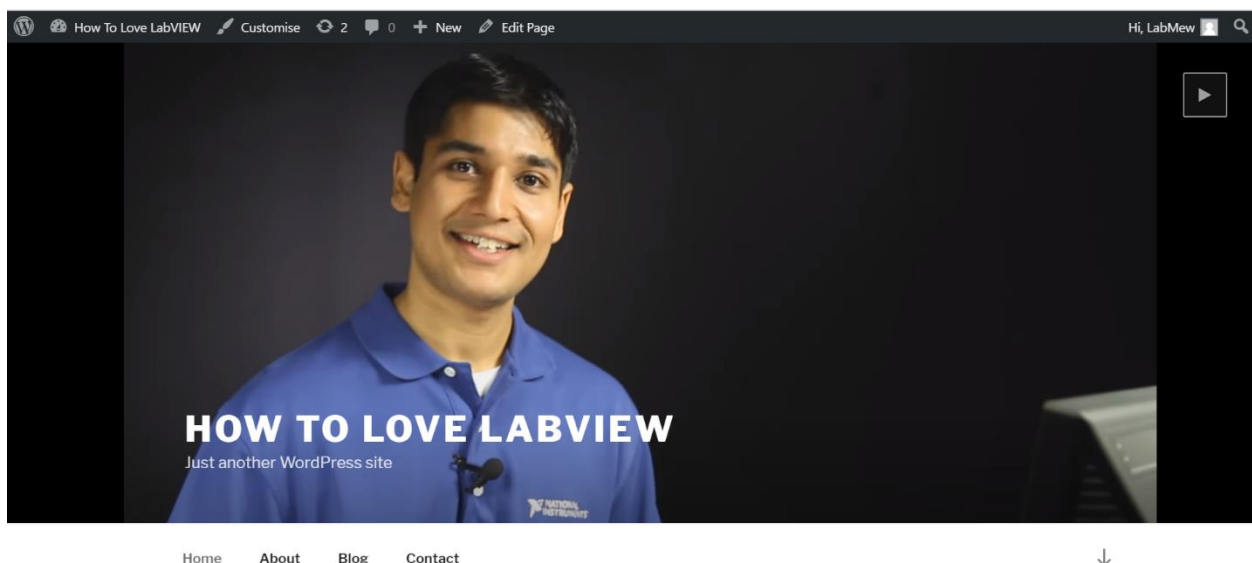
5. WordPress

The `wp-config.php` file must be created and customized.

MariaDB must be started to create a database for use with the WordPress application. See section **2. MariaDB** for details about the bug that prevents MariaDB from OPKG from starting with the normal commands.

After MariaDB is started on the target, `mysql -e` commands can be used to create the WordPress database.

When Apache and MariaDB are running (see section XXXX on how to start the LAMP services), use a web browser to connect to <http://<TARGET IP>:8081/wp-admin/install.php>, where <TARGET IP> is replaced with the IP address of your target, e.g. 10.1.128. This will guide you through the first-time configuration of your new WordPress blog. After the first-time configuration, use <http://<TARGET IP>:8081/wp-login.php> to access your WordPress blog.



To allow upload of content (pictures on blog, etc.) from a browser, you will have to modify the permissions on the `/home/admin/wordpress/wp-content/` folder to allow the web server process write access using `chown`.

6. Manually Start Services On Reboot of Target

Our Apache installation, as modified to run on port 8081, can run simultaneously with the NI Web-Based Configuration and Monitoring Service (WIF), which runs by default on port 80 if no self-signed certificate (SSL certificate) is configured on Apache. If Apache has been configured with a SSL certificate, there will be a conflict between these two web services on port 443.

A. Shut Down NI Web-Based Configuration and Monitoring

NI Web-Based Configuration and Monitoring service can be killed on startup. Use the following commands.

```
ps aux | grep Watchdog # Show PID number {NI WSD Watchdog} process.  
kill XXXX # where XXXX is the PID of the {NI WSD Watchdog} process on your target.
```

An alternative method is to use `top` to list all processes and find the {NI WSD Watchdog} process in the list to obtain PID.

The Watchdog process must be killed, or the WIF service will restart automatically.

B. Start Apache

Apache should be configured to run on startup if you have run the script, as it is added to startup services using the `update-rc.d` command. However, if it fails to start due to port conflict with WIF service, start Apache with the command `/etc/init.d/apache2 start` (if you have added it to startup scripts) or `/usr/local/apache2/bin/apachectl` (if you have not).

C. Start MariaDB

MariaDB is configured by default to run on startup, but may fail to start due to the bug described in section **2. MariaDB**. Start MariaDB manually with the command `mysqld`. This will put the terminal in an unresponsive state and a new session should be opened to continue sending commands to the target.

7. SSL Certificate and Google Geolocation Demo

Use OpenSSL to create a self-signed certificate for use with your cRIO Apache server. This is for demonstrative and development purposes only, a free or cheap SSL certificate should be used over self-signed for any commercially-used server.

This will allow usage of the Google Maps and Geolocation APIs in a page running on the Apache server. Users connecting to the page on the Apache server will be shown in a map where they are located.

How to Create and Install an Apache Self-Signed Certificate

<https://www.sslshopper.com/article-how-to-create-and-install-an-apache-self-signed-certificate.html>

Run the OpenSSL command in a new directory in the `apache2` folder called `ssl`.

```
mkdir /usr/local/apache2/ssl
cd /usr/local/apache2/ssl
openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout
crio_ssl.key -out crio_ssl.crt
```

Uncomment the following modules in `httpd.conf`:

```
LoadModule socache_shmcb_module modules/mod_socache_shmcb.so
LoadModule socache_memcache_module
modules/mod_socache_memcache.so
LoadModule ssl_module modules/mod_ssl.so
```

Edit `httpd-ssl.conf` to reflect the paths of the `.crt` and `.key` files. This config file can be found at `/usr/local/apache2/conf/extra/httpd-ssl.conf`.

Example: Google Maps API Geolocation Script

<https://www.sitepoint.com/working-with-geolocation-and-google-maps-api/>

Create a new `.html` document in the Apache `htdocs` folder and add the geolocation script. You may have to modify URL in the script to an `https` URL.

Restart Apache.

```
/etc/init.d/apache2 restart
```

File Encryption with GnuPG

1. Installation

To install `gpg` (GnuPG) 2.0.19 version use the following commands (in the `ssh` console):

```
opkg update
opkg install gnupg
```

*Note: if you would like to use this version of GnuPG from Linux shell and not LabVIEW you will need to install the Pinentry module that doesn't get installed through `opkg install` command. See Appendix A for installation instructions.

If you would like to install the classic version of GnuPG (1.4) you need to run the following commands to install this version from source:

```
wget ftp://ftp.gnupg.org/gcrypt/gnupg/gnupg-1.4.19.tar.bz2
tar xfv gnupg-1.4.19.tar.bz2
cd gnupg-1.4.19
./configure
make
make install
```

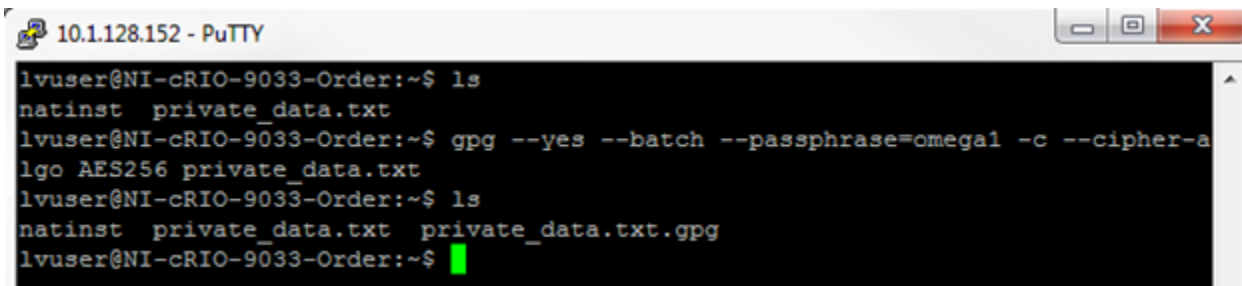
2. Getting Started

To encrypt a file with a symmetric cipher you can either use the Linux ssh console or do it in LabVIEW :

Encrypting a file through an ssh console:

To encrypt a file run the following command (assuming the file you're encrypting is /home/lvuser/private_data.txt):

```
gpg --yes --batch --passphrase=omega1 -c --cipher-algo AES256
private_data.txt
```

A screenshot of a PuTTY terminal window titled "10.1.128.152 - PuTTY". The terminal shows a user at the prompt "lvuser@NI-cRIO-9033-Order:~\$". The user runs "ls" and sees "natinst private_data.txt". Then they run "gpg --yes --batch --passphrase=omega1 -c --cipher-algo AES256 private_data.txt". After another "ls", they see "natinst private_data.txt private_data.txt.gpg". The prompt is then "lvuser@NI-cRIO-9033-Order:~\$" with a green cursor.

```
10.1.128.152 - PuTTY
lvuser@NI-cRIO-9033-Order:~$ ls
natinst private_data.txt
lvuser@NI-cRIO-9033-Order:~$ gpg --yes --batch --passphrase=omega1 -c --cipher-a
lgo AES256 private_data.txt
lvuser@NI-cRIO-9033-Order:~$ ls
natinst private_data.txt private_data.txt.gpg
lvuser@NI-cRIO-9033-Order:~$
```

GnuPG will create an encrypted file named private_data.txt.gpg in the same directory as your original file. Once the file is encrypted you can remove the original file:

```
rm private_data.txt
```

To decrypt the encrypted file use the following command:

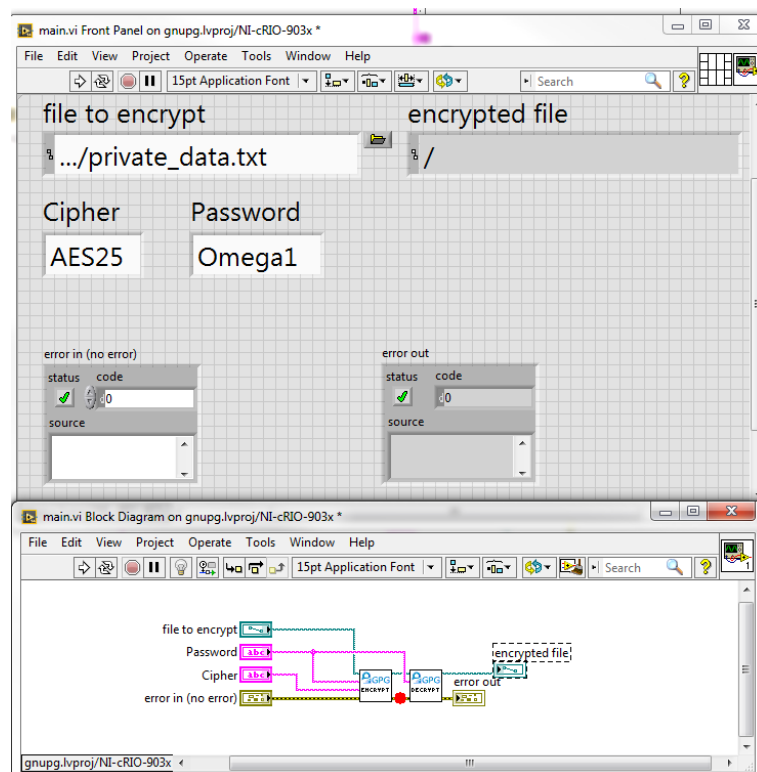
```
gpg --yes --batch --passphrase=omega1 private_data.txt.gpg
```

```
10.1.128.152 - PuTTY
lvuser@NI-cRIO-9033-Order:~$ ls
natinst private_data.txt.gpg
lvuser@NI-cRIO-9033-Order:~$ gpg --yes --batch --passphrase=omega1 private_data
.txt.gpg
gpg: AES256 encrypted data
gpg: encrypted with 1 passphrase
lvuser@NI-cRIO-9033-Order:~$ ls
natinst private_data.txt private_data.txt.gpg
lvuser@NI-cRIO-9033-Order:~$
```

Encrypting a file through LabVIEW:

To encrypt a file in LabVIEW (assuming the file you're encrypting is /home/lvuser/private_data.txt) :

1. Open gnupg.lvproj
2. Run main.vi



References:

1. <https://www.gnupg.org/gph/en/manual.html>
2. <https://www.gnupg.org/documentation/manuals/gnupg.pdf>
3. http://www.cryptnet.net/fdp/crypto/keysigning_party/en/keysigning_party.html
4. http://www.infosectoday.com/Articles/Intro_to_Cryptography/Introduction_Encryption_Algorithms.htm

Sharing files between cRIOs and Windows Machines

1. Installation

To install Samba use following command (in the ssh console):

```
opkg update
opkg install samba
```

2. Getting Started

To start using Samba you will need to create/modify `/etc/samba/smb.conf` for your specific network configuration. Instead of using the default `smb.conf` file that installs with samba it is usually easier to create one from scratch. To create a blank `smb.conf`:

```
mv /etc/samba/smb.conf /etc/samba/smb.conf.bak
touch /etc/samba/smb.conf
```

The above commands will create a blank `smb.conf` file that we can start editing. To edit the file:

```
cd /etc/samba
vi /etc/samba/smb.conf
```

The above commands will open the `smb.conf` file in the vi text file editor. Copy the following configuration information into the `smb.conf` file:

```
[global]

netbios name = cRIO-9034
workgroup = WORKGROUP
security = user
encrypt passwords = yes
smb passwd file = /etc/samba/smbpasswd
smb ports = 445 139
server signing = auto
interfaces = 10.1.128.1/255

[share]

comment = COMMENT
browseable = yes
valid users = admin
admin users = admin
```

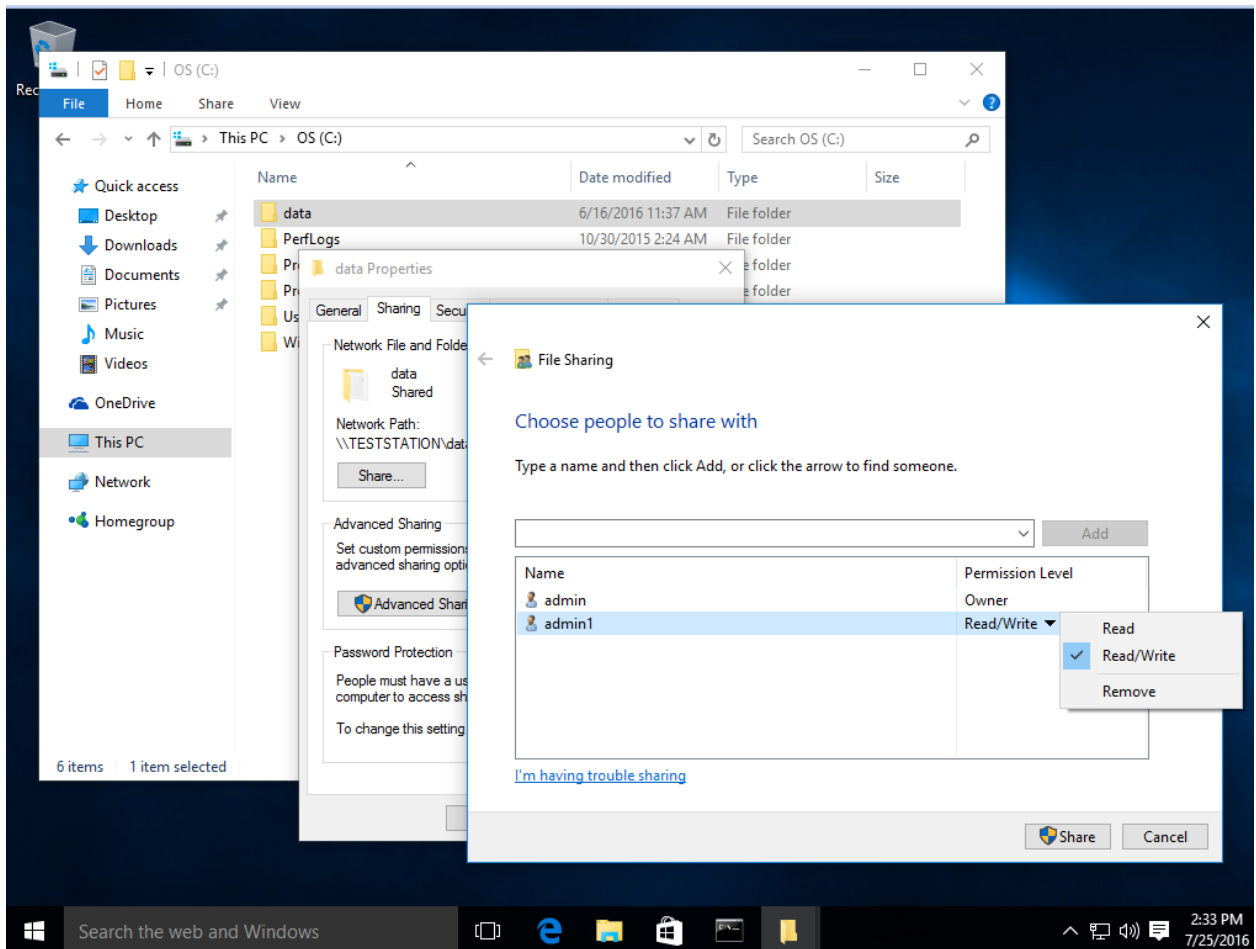
```
path = /home/admin/shared
writeable = yes
create mask = 0770
force create mode = 0770
locking = yes
```

Save the smb.conf file and restart the samba service for changes to take effect:

```
/etc/init.d/samba restart
```

Mounting a Windows shared folder on a cRIO

1. Create a shared folder on a Windows machine. The example below assumes that you have folder c:\data on your Windows machine and user "admin1" has Read/Write access to that folder



2. To mount this Windows shared folder in NI Real-Time Linux:

```
mkdir -p /mnt/data
```

```
mount -t cifs //10.1.128.113/data -o  
username=admin1,password=Password /mnt/data
```

The example above assumes that the Windows shared folder is located `//10.1.128.113/data` and it's accessing to a Windows user "admin1". You can start browsing the Windows shared folder by:

```
cd /mnt/data  
ls
```

Accessing a cRIO NI Real Time Linux shared folder in Windows

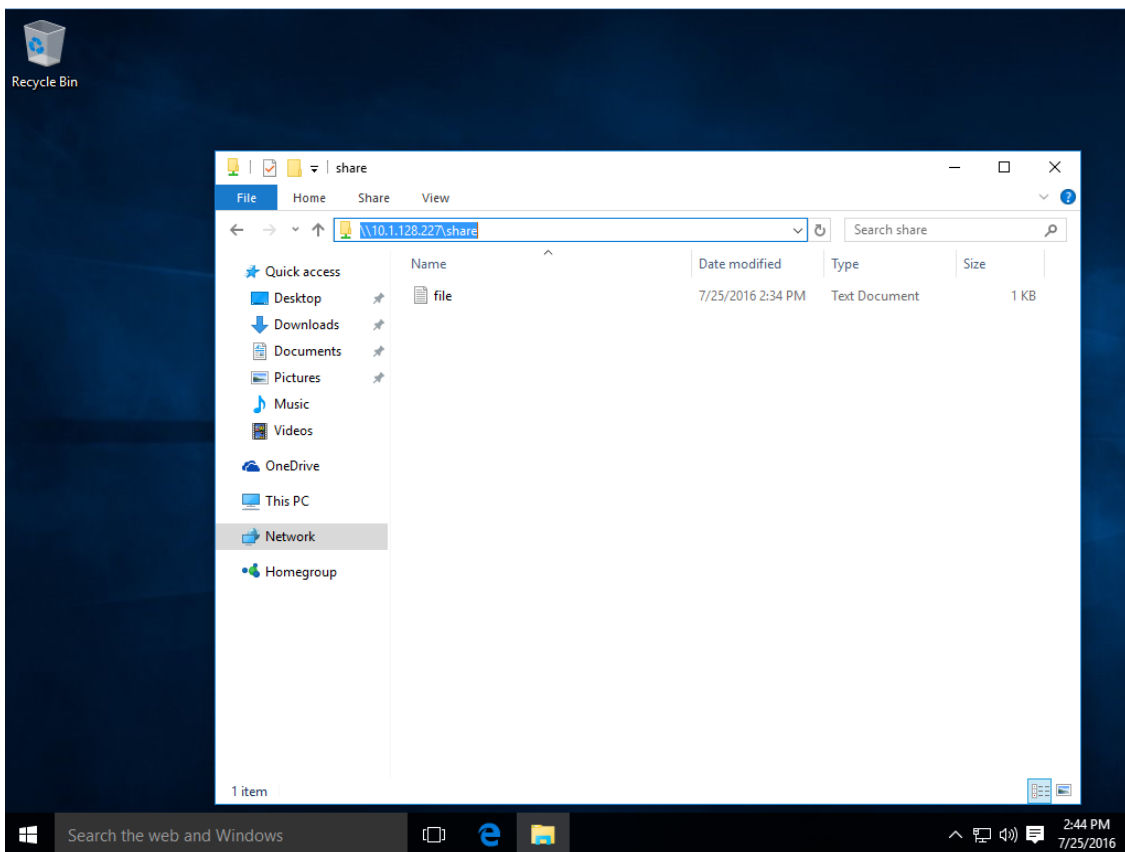
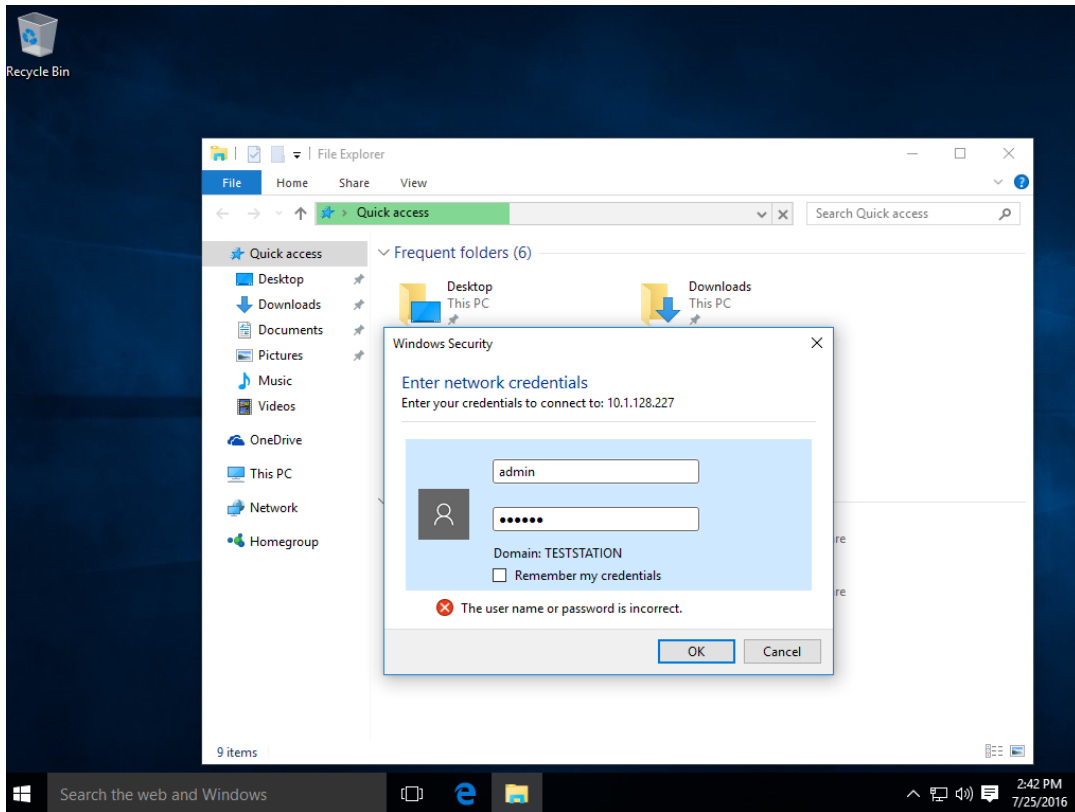
1. Create a shared folder on the cRIO:

```
mkdir /home/admin/shared
```

2. Since we've added this folder in the `[share]` section of the `smb.conf` file and specified that "admin" is the user that has read/write access to the folder we will need to add that user to samba:

```
smbpasswd -L -a admin  
smbpasswd -L -e admin
```

3. Once you've added and enabled the user "admin" to the list of users that can access the `/home/admin/shared` folder you can access the folder in Windows by typing `\\10.1.128.2287` (which is the IP address of the cRIO that has the shared folder) and logging in with the samba credentials specified in step 2.



References:

2. <https://www.samba.org/samba/docs/man/manpages/>
3. <https://www.linux.com/learn/tutorials/296391-easy-samba-setup>
4. <http://www.shellhacks.com/en/HowTo-Mount-Remote-Windows-Partition-Share-under-Linux>

Code Reuse (C/C++)

1. Background:

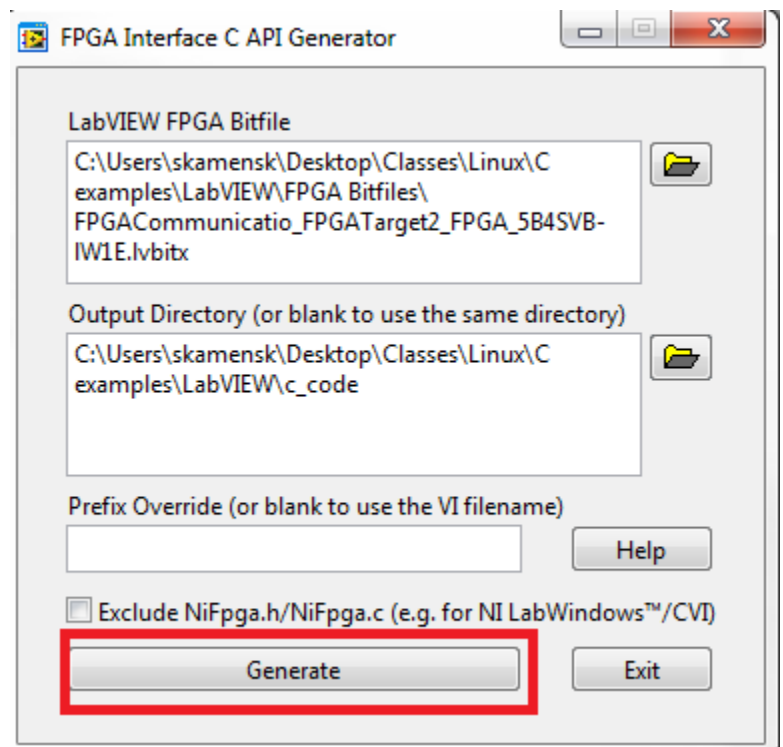
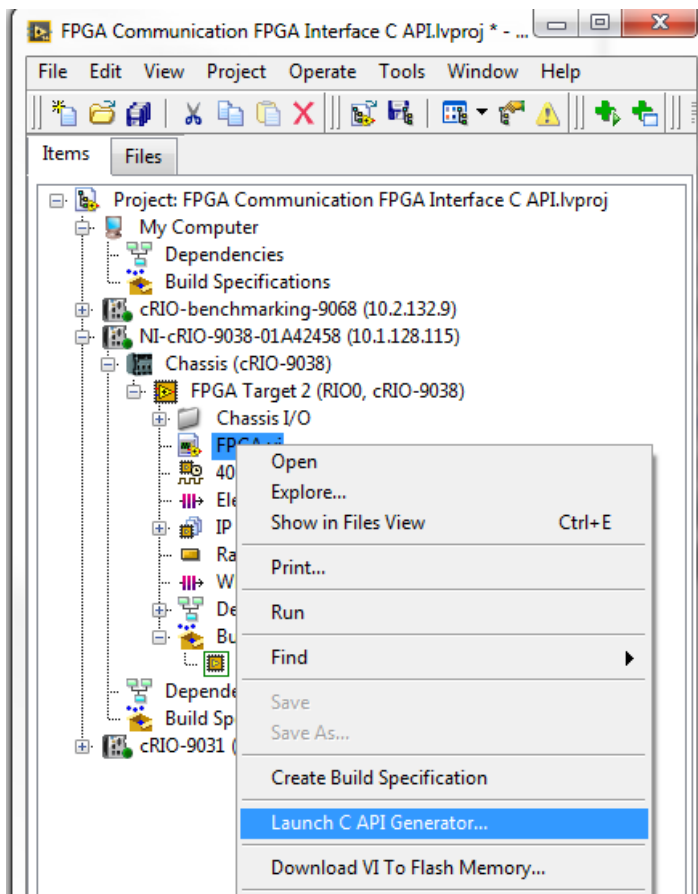
1. [C/C++ Embedded System Design Tools](#)
2. [Introduction to the FPGA Interface C API](#)

To run the C++ example code that demonstrates how to transfer data between FPGA and a C/C++ application on NI Linux Real-Time you have two options:

1. Follow instructions in this [example](#) and run the code through Eclipse
2. Compile and run the code on your NI Linux Real-Time target (demonstrated below)

In both cases you need to first generate C header files from a LabVIEW FPGA VI that will have the necessary C functions to access FPGA IO data in your C/C++ application:

1. Open FPGA Communication FPGA Interface C API.lvproj
2. Add you cRIO target and compile the FPGA.vi
3. Use FPGA C API Generator the generate C header files for FPGA code in the FPGA.vi:



Once C header files are generated follow the steps below to install GCC compiler and then compile and run the example code.

2. Installation

To install the GCC compiler and the tools necessary to compile and build the example code on the cRIO itself you will need to run the following commands in the ssh console:

```
opkg update
opkg install gcc gcc-symlinks
opkg install g++ g++-symlinks
opkg install cpp cpp-symlinks
opkg install libc6
opkg install libgcc-s-dev
opkg install binutils
opkg install binutils-symlinks
opkg install git
opkg install libstdc++-dev
opkg install make
```

3. Getting Started

Once you install GCC compiler you will need to copy the example code and C header files to the cRIO target:

1. Copy the generated files (see background section) as well as the Main.cpp to the NI Linux Real-Time target. The following files need to be copied to the cRIO (ex.
/home/admin/fpga-example/src)
..\FPGA_General_Communication\Main.cpp
..\LabVIEW\c_code\NiFpga.c
..\LabVIEW\c_code\NiFpga.h
..\LabVIEW\c_code\NiFpga_FPGA.h
..\LabVIEW\c_code\NiFpga_FPGA.lvbitx
2. Compile and build the example code:

```
gcc -x c++ Main.cpp NiFpga.c -lstdc++ -ldl -o FPGA_example
```

```
10.1.128.227 - PuTTY
admin@NI-cRIO-9034:~/fpga-example/src# ls
Main.cpp          NiFpga.h          NiFpga_FPGA.lvbtx
NiFpga.c          NiFpga_FPGA.h
admin@NI-cRIO-9034:~/fpga-example/src# gcc -O0 -g3 -Wall -c -fmessage-length=0 -o NiFpga.o NiFpga.c
admin@NI-cRIO-9034:~/fpga-example/src# g++ -O0 -g3 -Wall -c -fmessage-length=0 -o Main.o Main.cpp
admin@NI-cRIO-9034:~/fpga-example/src# g++ -o FPGA_example NiFpga.o Main.o -ldl
admin@NI-cRIO-9034:~/fpga-example/src#
```

Run the code

```
./FPGA_example
```

Code Reuse (Python)

1. Installation

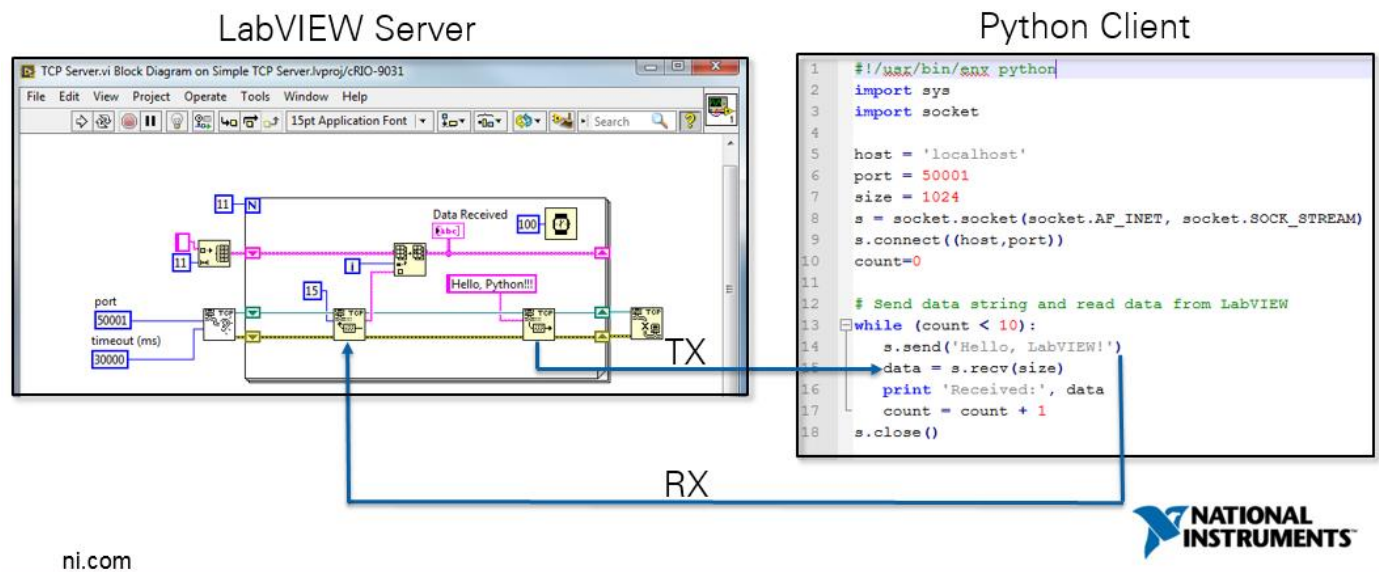
Before proceeding with installation make sure you have the packages from [Appendix B](#) installed on the cRIO. To install Python download the source and install using the following commands :

```
wget --no-check-certificate https://www.python.org/ftp/python/2.7.12/Python-2.7.12.tgz
tar -xvf Python-2.7.12.tgz
cd Python-2.7.12
./configure
make
make install
```

```
10.1.128.227 - PuTTY
admin@NI-cRIO-9034:~/Python-2.7.12# ls
Demo          Makefile.pre      Python           config.sub
Doc           Makefile.pre.in   README          configure
Grammar       Misc              RISCOS          configure.ac
Include       Modules           Tools           install-sh
LICENSE       Objects           aclocal.m4      pyconfig.h
Lib           PC                config.guess     pyconfig.h.in
Mac           PCbuild           config.log       setup.py
Makefile      Parser            config.status
admin@NI-cRIO-9034:~/Python-2.7.12# ./configure
```

2. Getting Started

Let's look at a simple example of passing data between LabVIEW and Python using a TCP socket:



1. Connect to your cRIO through SSH
2. Copy or create `python_tcp_client.py` script

```
10.1.128.228 - PuTTY
admin@cRIO-9031:~# ls
Python-2.7.10      code      python_tcp_client.py
Python-2.7.10.tgz  compiler_install.sh  shared
admin@cRIO-9031:~#
```

```
10.1.128.228 - PuTTY
#!/usr/bin/env python

"""
A simple echo client
"""

import sys
import socket

host = 'localhost'
port = 50001
size = 1024
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((host,port))
count=0
data='Hello, LabVIEW!'

# Check the size of the data(in bytes) you're sending
#data_size = len(data);
#print >>sys.stderr, 'Data size:', data_size

# Send data string and read data from LabVIEW
while (count < 10):
    s.send('Hello, LabVIEW!')
    data = s.recv(size)
    print 'Received:', data
    count = count + 1
s.close()
python tcp_client.py
```

3. Open Simple

TCP Server.lvproj and Run TCP Server.vi

4. Run `python_tcp_client.py`

The screenshot displays the LabVIEW environment. The top window, titled "TCP Server.vi on Simple TCP Server.lvproj/cRIO-9031", shows a menu bar (File, Edit, View, Project, Operate, Tools, Window, Help) and a toolbar. Below it, the "TCP Server.vi Block Diagram on Simple TCP Server.lvproj/cRIO-9031" is visible. The block diagram includes a "port" input set to 50001, a "timeout (ms)" input set to 30000, and a "Data Received" output. The diagram features several "TCP" blocks and a "Hello, Python!!" message box. To the right of the block diagram, a "Data Received" list shows multiple entries of "Hello, LabVIEW!".

References:

1. <http://pymotw.com/2/socket/tcp.html>
2. <http://ilab.cs.byu.edu/python/socket/echoserver.html>

Code Reuse (node.js)

1. Installation

Before proceeding with installation make sure you have the packages from Appendix B as well as python (see previous section) are installed on the cRIO. To install node.js download the source from <https://nodejs.org/dist/v4.4.7/node-v4.4.7.tar.gz> and extract the contents of the archive into /home/admin/node-v4.4.7:

```
wget https://nodejs.org/dist/v4.4.7/node-v4.4.7.tar.gz
tar -xvf node-v4.4.7.tar.gz
cd node-v4.4.7
./configure --without-ssl
find ./src/node.cc -type f -exec sed -i 's/SSL2_ENABLE = true;\\/\\/SSL2_ENABLE = true;/g' {} \;
find ./src/node.cc -type f -exec sed -i 's/SSL3_ENABLE = true;\\/\\/SSL3_ENABLE = true;/g' {} \;
make
make install
```

2. Getting Started

Let's look at a simple example from nodejs.org of how we can use node.js to setup a simple webserver:

1. Create an example.js file and copy the following contents into the file:

```
//This simple web server written in Node responds with "Hello World" for every request.
// change the ip address 10.1.128.228 to the ip address of your cRIO
```

```
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello World\n');
}).listen(8081, '10.1.128.228');
console.log('Server running at http://10.1.128.228:8081/');
```



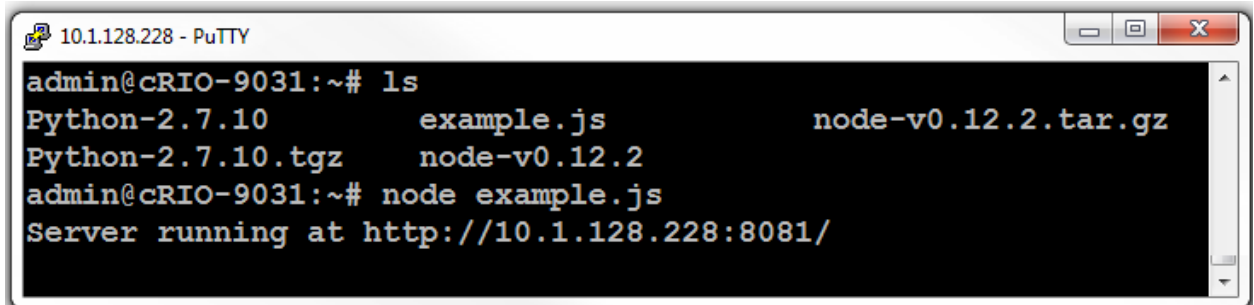
```

1  //This simple web server written in Node responds
2  //with "Hello World" for every request.
3
4  var http = require('http');
5  http.createServer(function (req, res) {
6      res.writeHead(200, {'Content-Type': 'text/plain'});
7      res.end('Hello World\n');
8  }).listen(1337, '127.0.0.1');
9  console.log('Server running at http://127.0.0.1:1337/');

```

2. Run the program:

```
node example.js
```



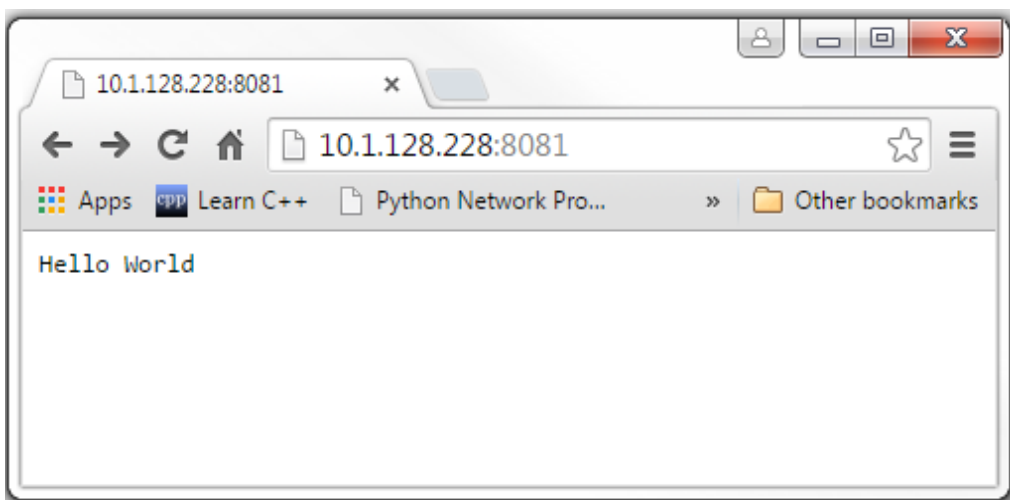
A terminal window titled "10.1.128.228 - PuTTY" shows the following commands and output:

```

admin@cRIO-9031:~# ls
Python-2.7.10      example.js      node-v0.12.2.tar.gz
Python-2.7.10.tgz  node-v0.12.2
admin@cRIO-9031:~# node example.js
Server running at http://10.1.128.228:8081/

```

3. Navigate to the ip address of your cRIO:



References:

1. <http://nodejs.org>

APPENDIX A

Installing Pinentry module for GnuPG from source:

1. Install the following packages in order to be able to compile the pinentry module from source:

```
opkg update
opkg install bzip2
opkg install gcc gcc-symlinks
opkg install g++ g++-symlinks
opkg install cpp cpp-symlinks
opkg install libc6
opkg install libgcc-s-dev
opkg install binutils
opkg install binutils-symlinks
opkg install git
opkg install libstdc++-dev
opkg install make
opkg install wget
opkg install automake
opkg install autoconf
opkg install libncursesw5
opkg install libqtcore4
opkg install libqtcore-dev
opkg install libncurses5
```

2. Download the pinentry module and compile it:
(The commands below need to be executed in the following order)

```
wget ftp://ftp.gnupg.org/gcrypt/pinentry/pinentry-0.9.4.tar.bz2
tar xfv pinentry-0.9.4.tar.bz2
cd pinentry-0.9.4
./configure
mkdir ncursesw
cp /usr/include/curses.h ncursesw
make
make install
cp /usr/local/bin/pinentry /usr/bin
```

APPENDIX B

In order to be able to compile software in this tutorial from source you need to install the following packages:

```
opkg update
opkg install gcc gcc-symlinks
opkg install cpp cpp-symlinks
opkg install g++ g++-symlinks
opkg install bzip2
opkg install bzip2-dev
opkg install libbz2-dev
opkg install libc6
opkg install libgcc-s-dev
opkg install binutils
opkg install binutils-symlinks
opkg install git
opkg install libstdc++-dev
opkg install make
opkg install wget
```