**Linux Terminal SSH Communication Code Log**

Last Modified: February 15, 2016

***Jonathan Watts, Undergraduate Research Assistant***

***(Please Add Additional Contributors Here)***

*Malware Visualization, Distributed Analytics and Security Institute*

*High Performance Computing Collaboratory, Mississippi State University*

**Objective:** This document provides of a brief overview of how to communicate with a remote Linux terminal using a SSH connection and C# script. It also represents a log of our progress as the code is further modified.

**Introduction:** Due to security limitations of Telnet, we have chosen to use an SSH connection moving forward. Therefore, we must restructure all of our Linux terminal communication code accordingly. We have found that the most complete C# library available for creating and utilizing an SSH connection is SSH.NET. For more information on this library, please see the following links:

Home Page: [sshnet.codeplex.com](http://sshnet.codeplex.com)

Source Code / Documentation: [sshnet.codeplex.com/releases/view/120565](http://sshnet.codeplex.com/releases/view/120565)

Using this library, we have been able to successfully connect and communicate with a remote Linux terminal. However, the difficulties occur when modifying the code for use in a Unity environment. (The C# scripts will be triggered by an interaction in the environment.) The original code operates by receiving user input through a Windows console. The challenge is sending and receiving the necessary information purely through code. However, this certainly can be accomplished with a little work. This document will continually track the progress of these efforts and is subject to change. The secondary objective is that the documentation of these endeavors will one day serve as a guide for anyone who desires to accomplish a similar task.

**Progress Log:**

**2/12/16:** The current code’s functionality depends on an infinite loop, which continually checks for data from the remote terminal. The receipt of this data triggers an event and executes a function to perform an asynchronous read and display that data to the Windows console. After the read is complete, the user is given the opportunity to provide input. This segment of the code is shown below:

*// This loop continually checks for data from the remote terminal*

while(true)

{

*// when data is received, the StartAsyncRead function is called*

stream.DataReceived += StartAsyncRead;

*// user input from the console is stored in the string variable cmd*

cmd = Console.ReadLine();

*// the command is then sent to the remote terminal and executes*

stream.WriteLine(cmd);

}

A modification that needs to be made immediately is to eliminate any code that requires use of the Windows console. It is not practical to access the console while inside the Unity environment. Instead, we would prefer if the same tasks could be accomplished by the passing of a variable, or some other code functionality. In this case, we need to identify the command needed for a particular interaction with the terminal, store that command as a string variable, and pass that variable into the desired function so that it may be sent to the terminal and executed.

One aspect of this infinite loop that we do like is that it can run autonomously in the background while other code is running. Essentially, I hope to design a multithreaded application where other C# scripts can interact with the above loop as needed. This will most likely require the use of the BackgroundWorker component. For documentation of this class, please see the following link:

[msdn.microsoft.com/en-us/library/system.componentmodel.backgroundworker.aspx](http://msdn.microsoft.com/en-us/library/system.componentmodel.backgroundworker.aspx)

This approach is advantageous for our project because it allows us to isolate our interactions with the terminal. If in the future we are required to analyze a different dataset, we would only be required to make a minimal number of changes to the code.

**2/15/16:** Before implementing the multithreaded approach, I would like to develop a system to retrieve the data from the stream buffer in its entirety. I was able to parse the output to remove some of the junk characters, but it is still showing the output twice in some cases. (More exceptions will need to be made as we test more commands and add more functionality.) The code I added is shown below:

line = line.Replace("[0K\ r", "");

line = line.Replace("[0m", "");

line = line.Replace("[33m", “”);

*// removes leftward arrow character -- does not show up in Visual Studio*

line = line.Replace("", “");

The last line may look like it does nothing, but it actually removes leftward arrow characters from the string. This character cannot be displayed in Visual Studio, so it appears the same as an empty char.