Improve Cuckoo's Ability Of Analyzing Network Traffic

Abdulellah Abdulaziz Al-Saheel
Supervised By
Claudio Guarnieri
Honeynet Organiztion
Google Summer Of Code 2012

August 17, 2012

Contents

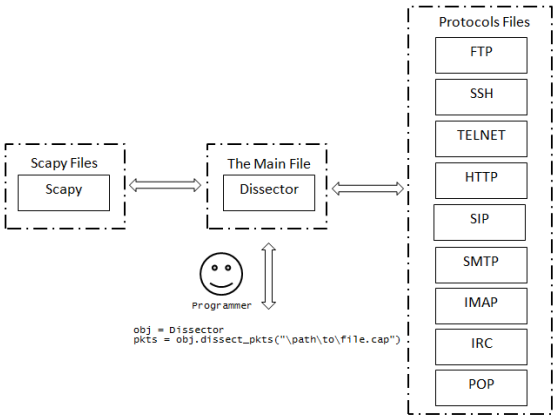
Chapter 1

1.1 Introduction

Cuckoo Sandbox is an Automated Malware Analysis developed by Claudio Guarnieri, mainly Cuckoo is a lightweight solution that performs automated dynamic analysis of provided Windows binaries. It is able to return comprehensive reports on key API calls and network activity. This documentation is introduce you a library which processes the network files (PCAP files or Packet-Capture files) and return back a report of the result. This library dissect packets fields and extract the most possible extent of information out of network packets, it also aware of tcp reassemblingn not just that it can recover the downloaded files for http, ftp and the sent emails by smtp.

1.2 Description

This library depend on Scapy library. The supported protocols in this library are: TCP, UDP, ICMP, DNS, SMB, HTTP, FTP, IRC, SIP, TELNET, SSH, SMTP, IMAP and POP. Even that the first five protocols were supported by Scapy they have been interfaced by this library. This figure demonstrates the transparent structure of the library:



1- The main component in this library which is dissector is responsible of receiving a path to pcap file and send back a dictionary of the supported protocols which holds the dissected packets. Also this component is the one who specify how to represent the data and also it is the responsible of importing Scapy classes and the library classes. Also it preprocesses the tcp sequence numbers and implements the tcp reassembly.

2- The protocols files, each file has one or more classes which responsible for dissecting the corresponding protocol packets.

3- There are set of Scapy classes have been used in this library which are Packet class inherited by "Protocols classes", and Field class which inherited by "Fields classes" and it does use rdpcap which takes a path to pcap file and returns back a list of packets.

1.3 General Protocol File Structure

For any future development no need to go deep in Scapy since in this library I didn't use advanced features of Scapy, so I am going to introduce you the simplest (pseudo code) form of a protocol file structure I followed in this library annotated with some comments:

```
class FTP (Packet):
'''
 since it inherits Packet class we can classify this class as "Protocol Class".
'''
    name = "ftp" # name of the protocol

    fields_desc = [FTPField("command", "", "H"), StrField("argument", "", "H")]
    '''
    fields_desc is a list used to describe the protocol fields, even though
    you will not retrieve the fields values from fields_desc but you
    will do from another list called fields.
    '''

bind_layers(TCP, FTP, sport=21)
'''
bind_layers used as glue in Scapy to sendpackets with the specified port
number to your protocol class.
'''

bind_layers(TCP, FTP, dport=21)
'''
the same here, but here it uses the destination port not the source port.
'''
```

Are we done of the protocol file? well, Not yet. As you see in the previous code in fields_desc we have used a class named FTPField and this class is "Field Class" which means in either way it should inherits Field class of Scapy, the other class StrField this has the same thing it inherits Field class but it is predefined by Scapy. Now let us have a look at FTPField class.

```
class FTPField(Field): # our Field class
    def getfield(self, pkt, s):
        '''
        dissecting goes here. The bytes stream stored in s, this method
        needs to "return" two values, the value for the current field and
        the remaining of the stream which will be passed to the next field
        in the fields list.
        '''

        value = ..... # part of the stream s
        remaining = s  value # collecting the remaining of the stream s

        return remaining, value
        '''
        remaining will be passed to the next field whereas
        value will be assigned to the current field.
        '''
```

1.4 Protocols Details and Notes

Different protocols have different properties especially when you go in details. So here I am going to lists the different characteristics and features of the implemented protocols.

Protocol	Notes \ Characteristics
FTP	Recovers the downloaded files. Returns strings and binary data encoded with base64. If PASSIVE command issued negotiated port number will be used for transferring the data.
SSH	Returns strings and binary data encoded with base64. It has its own tcp reassembly implemented internally this is because ssh is so sensitive in this and because of its nature of tunneling.
TELNET	Returns strings and binary data encoded with base64.
HTTP	Recovers the downloaded files. Returns strings and for message-body field its value comes encoded with base64.
SIP	Returns strings and for message-body field its value comes encoded with base64. It is so similar to http.
SMTP	Recovers the sent emails. Returns strings and binary data encoded with base64.
IMAP	Returns strings
POP	Returns strings
IRC	Returns strings

1.5 Usage

Here you will see simple use of this library. Let us have our file usedissector.py as follows:

```
from dissector import *

dissector = Dissector() # instance of dissector class
pkts = dissector.dissect_pkts("/path/to/file.cap")
'''
sending the pcap file to be dissected
'''

print(pkts) # print the packets
```

the output will be similar to this:

```
{'ftp': [....], 'http': [....], ....}
```

1.6 Downloaded Files Recovery

I have wrote a dedicated section for the files recovery to state how this feature works for http, ftp and smtp. All of the protocols will create a directory named downloaded in the current working directory (CWD) to store the recovered files. in case that you want to change the default and want to store the recovered files in another directory you have to send a path to change_dfolder just like this:

```
from dissector import *

dissector = Dissector() # instance of dissector class
dissector.change_dfolder("\root\Desktop\")
'''
now the downloaded files will be stored on the desktop
'''

pkts = dissector.dissect_pkts("/path/to/file.cap")
'''
sending the pcap file to be dissected
'''
```

for http it takes the file name from the start line of the http request, so if another file has the same name in the specified directory or the name has some special characters then a random name will be generated. the same apply for ftp which takes the file name from RETR command. whereas smtp just gives the file a random name.

1.7 Source Code

the source code of this library is on github:
$ git clone https://github.com/cssaheel/dissectors.git