DESIGN AND IMPLEMENTATION OF NETWORK COLLECTOR

Jaroslav Bosela
Master Degree Programme (2), FEEC BUT
E-mail: xbosel00@stud.feec.vutbr.cz

Supervised by: Vaclav Oujezsky
E-mail: oujezsky@feec.vutbr.cz

Abstract: This article presents a part of our project focused on developing a network collector and analyzer. It is a Python application used to collect and analyze NetFlow version 9 messages. With this article, an underlying schema and code of the application are described.

Keywords: Analyzer, collector, network, NetFlow, python

1 INTRODUCTION

In today's world, it's essential to analyze and collect network traffic. Such analysis gives us important information, which is needed for various functions and purposes. In the early stage of Internet networks were analyses primarily used for better routing, network speed rising, and improvement for the whole transmission. With the enormous growth of users and transmission point in today's time, the need for more extensive use of this information has risen. Analysis with this information has begun to use in detection of security risk, expose weak spots in network and network behavior as the entire thing. For this analysis, we use export network protocols, such a NetFlow [1] and IPFIX (Internet Protocol Flow Information Export) [2], which are most common in use. These protocols help network administrators and experts to detect vulnerable network nodes, their abnormal behavior, or check atypical traffic run. Export network protocols also provide us information about inner and outer incidents, improve the whole network process, and take care of a balanced load of the network. This article focuses on Cisco's network export protocol named NetFlow, in version 9 [3].

The output of NetFlow is a flow record. The most recent version of the NetFlow flow record format is NetFlow version 9. The characteristic of the NetFlow Version 9 format is that it is template-based. A template defines a collection of fields, with corresponding descriptions of structure and semantics [4]. The principle of reading and parsing data depending on the template is the most important work on the collector, described in the following section.

2 NETWORK COLLECTOR

The NetFlow network collector is being developed in Python version 3. Currently, support for NetFlow messaging version 9 is under development. The development of NetFlow version 1 and 5 processing is now complete [5]. The program listens for incoming NetFlow reports on individually selected port. The port number can be changed. The port number 4710 is selected by default. The entire socket consists of all interfaces and addresses on the device and of the selected port number.

The program automatically stores information about a traffic to the sqlite3 database file. The database file is deleted by default from the program's beginning. The preservation of historical data in the database can be changed with the configuration file.

2.1 The processing of NetFlow 9

Designing and processing of NetFlow version 9 is considerably more difficult. In this version, dynamic templates are used to determine what will be contained in custom NetFlow messages. The code is designed based on the standard and is structured into classes for clarity. The NetFlow header is constant and includes information about the NetFlow version and templates, Figure 1.

```
    class Header
        def __init__(self,data):
            pack = struct.unpack("!HHIIII", data[:self.length])
            self.version = pack[0]
            self.count = pack [1]
            self.uptime = pack [2]
            self.timestamp = pack[3]
            self.sequence = pack [4]
            self.source_id = pack [5]
```

Figure 1: Class for processing the packet header.

```
    class Data_Flow_Set:
        def __init__(self, data, template):
            pack = struct.unpack("!HH", data[:4])
            self.template_id = pack[0]
            self.length = pack[1]
            self.flows = []
            offset = 4
            template = templates[self.template_id]
                padding_size = 4 - (self.length \% 4)

    class Template_Field:
        def __init__(self, field_type, field_length):
            self.field_type = field_type
                self.field_length = field_length

    class Template_Flow_Set:
        def __init__(self, data):
            pack = struct.unpack("!HH", data[:4])
            self.flowset_id = pack[0]
            self.length = pack[1]
            self.templates = {}
            offset = 4

    class Export_Packet:
        def __init__(self, data, templates):
            self.header = Header(data)
            self.templates = templates
            self.flows = []
            offset = 20
```

Figure 2: Classes for processing the templates.

The `Data_Flow_Set` class, Figure 2, defines the sequence of data from the packet, which needs to be unpacked and looked into it. The `template_id` is filled, with an identifier of the template used in sequence, the `length` field with a length of flow set, other fields are filed as RFC (Request for Comments) definition says for format padding. The `template` field is a definition template by identifier from the previous field and is used in the following class. The `Template_Field` class defines type and length from template definitions, also the next defined `Template_Flow_Set` class brings flow set template, which can be used for carrying other templates used in traffic flow.

```
import socketserver
import NetFlowV9_collector
import ExportPacket

def get_server(cls, host, port):
    logging.info("Listening on {}:{}".format(host, port))
    server = socketserver.UDPServer((host, port),cls)
    return server
```

Figure 3: Algorithm to run the NetFlow collector.

The `Export_Packet` class describes the version of a packet that needs to be imported into the main file, where it is used for export to final data. This class contains header data, template, and flow data from previous classes.

The presented code introduces the main important parts of the classes and imports from the main file, as NetFlow collector, socket server, which is used for listening on ports, and export packet with the template, header, and flow data coming from exporter to collector.

The code presented in Figure 3 is used to start the socket listener on UDP (User Datagram Protocol) port and to get exported data, the length of these data, source ports, and IP (Internet Protocol) address.

3 CONCLUSION

Within this article, a network collector has been introduced that is being developed in Python to enable the processing of NetFlow messages in version 9. The basic concept of the network collector and its properties and possibilities were introduced in the paper. Once the collector is complete, a program extension will be developed to implement its own algorithms for detecting traffic anomalies.

ACKNOWLEDGEMENT

Research described in this paper was financed by the grant of the Ministry of the Interior of the Czech Republic, Program of Security Research, VI20192022135, PID VI3VS/746 “Deep hardware detection of network traffic of next generation passive optical network in critical infrastructures”.

REFERENCES

[1] “Introduction to cisco ios netflow - a technical overview,” 2012. [Online]. Available: http://www.cisco.com

[2] “Specification of the ip flow information export (ipfix) protocol for the exchange of flow information,” 2013. [Online]. Available: https://tools.ietf.org/html/rfc7011

[3] J. Bosela, “Implementation of internet protocol in python,” Master’s thesis, Brno University of Technology, 2019.

[4] “Cisco systems netflow services export version 9,” 2004. [Online]. Available: https://www.ietf.org/rfc/rfc3954.txt

[5] “Gdp - netflow collector,” 2016. [Online]. Available: nsr.utko.feec.vutbr.cz/software.php