IPDiff – Detecting IP Traffic Changes

Mbuku Tunga Ditutala

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisor: Prof. Doutor Miguel Nuno Dias Alves Pupo Correia

Examination Committee

Chairperson: Prof. Doutor Luís Manuel Antunes Veiga

Supervisor: Prof. Doutor Miguel Nuno Dias Alves Pupo Correia

Members of the Committee: Prof. Doutor Paulo Rogério Barreiros d'Almeida Pereira

July 2018

Acknowledgements

First of all, I thank to Jesus Christ, for keeping me healthy, which allowed me to get to this stage of my life.

In sequence, I would like to thank my beloved wife Dulce Ditutala for all you have been through, replacing me in many circumstances, acting as father and mother for our children – the 4Rs. Thanks to Romélia, Ricardo, Raquel and Renata for supporting me even being apart from you for this quite long period of time.

I also would like to thank my father João Ditutala Mose for inspiring me confidence and integrity. Thank you. I Thank to my brothers because you have been helping me along this entire journey of life among many difficulties. Thank you all.

All this is only possible due to the scholarship sponsored by Omnen Intellegenda S.A. So I thank the Omnen Intellegenda's board of directors.

I could not forget all of you whom crossed my way in TÉCNICO. I thank everyone, specially to Pushparaj Motamari and Sara Rodrigues, for your support in labs classes.

I want to pay my gratitude to DEI staff, for their patience and support when I needed. Special thank to Professors Jose Monteiro and Ricardo Chaves for welcoming me on their courses even having arrived late. I also would manifest my appreciation to Professor David Matos. You inspired me!

Lastly, I would like to say a special and big thank you to my supervisor, Professor Miguel Pupo Correia for your guidance, insights, helpful comments and for challenging me to write this thesis in english, which certainly have broaden my english writing skills. Thank you master.

Abstract

Network management software is very important for network operations and for network services delivery. Understanding network traffic usage is of great value when business decisions have to be taken. Network managers use software to keep track of network events. Network events may occur due to many reasons. But when there is an event, network management software should be able to provide meaningful information to the network administrator. So any good network management software should be able to draw attention to what is really happening. Much research has already addressed network traffic detection and classification. However much effort is still being done towards providing on time meaningful information to network administrators. This thesis addresses the problem of network traffic change detection and classification. The objective of the IPDiff tool is to compare network traffic of two time periods and to display the network traffic and the differences found. To evaluate IPDiff, experiments were conducted over a dataset of 5GB of network flows collected at the Los Alamos National Laboratory network. The experimental results show that IPDiff is capable of comparing and detecting differences on flows at different time periods.

Keywords: Traffic Change, Traffic Classification, Network Security, NetFlow, SiLK

Resumo

O software de gestão de rede é muito importante para a operação da rede e para a entrega de serviços de rede aos seus utilizadores finais. Para se tomarem decisões eficientes, a informação sobre o uso da rede é de capital importância. Os administradores de redes usam software para acompanhar todas as ocorrências na rede. Muita investigação e muito software já abordam a detecção e a classificação do tráfego na rede. No entanto, muito esforço ainda está sendo feito no sentido de fornecer em tempo útil informações significativas aos administradores de rede. Esta tese aborda o problema da detecção de alteração e classificaçao do tráfego de rede. O IPDiff tem como propósito comparar tráfego de dois periodos de tempo e visualizar o tráfego de rede e as diferenças detectadas. Para avaliar o IPDiff foram realizadas diversas experiências sobre os fluxos, num total de 5GB, coletados na rede do Los Alamos National Laboratory. Os resultados experimentais mostraram que o IPDiff foi capaz de comparar e detectar diferenças nos fluxos para os diferentes intervalos de tempo analisados.

Palavras-Chave: Alterações no Tráfego, Classificação do Tráfego, Segurança de Redes, NetFlow, SiLK

Contents

List of Tables

List of Figures

List of Acronyms

API Application Programming Interface.

AS Autonomous System.

ASN Autonomous System Number.

ASN.1 Abstract Syntax Notation.

BGP Border Gateway Protocol.

BPF Berkeley Packet Filter.

CERT Computer Emergency Readiness Team.

CMIP Common Management Information Protocol.

CORBA Common Object Request Broker Architecture.

DBB Distributed Big Brother.

DME Distributed Management Environment.

DNM Distributed Network Management.

DOC Distributed Object Computing.

DoS Denial of Service.

EJB Enterprise Java Beans.

FCAPS Fault, Configuration, Accounting, Performance and Security.

HTML Hypertext Markup Language.

HTTP Hypertext Transfer Protocol.

HTTPS Hypertext Transfer Protocol Secure.

IAB Internet Activities Board.

IANA Internet Assigned Numbers Authority.

ICMP Internet Control Message Protocol.

IDE Integrated Development Environment.

IDS Intrusion Detection System.

IETF Internet Engineering Task Force.

IP Internet Protocol.

IPFIX IP Flow Information Export Protocol.

ISP Internet Service Provider.

ITU-T International Telecommunication Union -Telecommunication Standardization Sector.

Java EE Java Enterprise Edition.

JPA Java Persistence API.

JSF JavaServer Faces.

JSON JavaScript Object Notation.

MbD Management By Delegation.

MIB Management Information Base.

ML Machine Learning.

MRTG Multi Router Traffic Grapher.

MVC Model View Controller.

NetSA Network Situational Awareness.

NIDS Network Intrusion Detection System.

NMS Network Management System.

NOC Network Operation Center.

OID Object Identifier.

OO Object Oriented.

ping Packet Internet Groper.

PRTG Paessler Router Traffic Grapher.

REST The Representational State Transfer.

RMON Remote Monitoring Protocol.

RPC Remote Procedure Call.

RRD Round Robin Database.

SCTP Stream Control Transmission Protocol.

SIFT Security Incident Fusion Tools.

SiLK System for Internet Level Knowledge.

SMI Struture for Management Information.

SNMP Simple Network Management Protocol.

SSH Secure Shell Protocol.

TCP Transmission Control Protocol.

TCP/IP Transmission Control Protocol/Internet Protocol.

TLS Transport Layer Security.

TTL Time-to-live.

UDP User Datagram Protocol.

UML Unified Modeling Language.

VM Virtual Machine.

WBEM Web-Base Enterprise Management.

XML Extensible Markup Language.

YAF Yet Another Flowmeter.

Chapter 1

Introduction

This first chapter starts by introducing the importance of the topic addressed by this thesis for the organizations. The chapter then presents a brief overview on the literature review related to the topic. Then the aim of the thesis is presented and the chapter ends by giving an outline of the remaining thesis chapters.

1.1 Motivation

Network traffic usage is an important and strategic topic to business processes. This awareness reduces network vulnerabilities thus minimizes network outages and allows efficient network operation. Improvements in network management and operation lower operational costs, leading to higher customer satisfaction and higher business revenues. With this in mind the proposed approach to network traffic comparison based on differential flows will contribute by introducing a novel methodology and a tool called IPDiff which can minimize computational performance needed to process network flows. Also by detecting malicious new flows, IPDiff will increase network security awareness, helping to the impact of network attacks when they occur.

1.2 Overview

The current internet users' needs are driving the production and deployment of more and more new applications. Moreover, the improper usage of the internet poses many security concerns [1, 2]. The growth of the internet has given rise to problems to information security. One of the fastest growing fields in order to solve those problems is network traffic monitoring. Many network applications publicly accessible use a port assigned by Internet Assigned Numbers Authority (IANA) [3].

The initial approach on solving network management problems was the development of network monitoring tools, such as ping, traceroute, and the Multi Router Traffic Grapher (MRTG) [4–7]. As more and more new applications were developed, port reuse became necessary and this has led to very challenging security issues since monitoring could no longer rely on ports because a port could be used by different applications on same network device. The development of those new applications and the emergence of increasingly specialized new forms of attacks has led to the development of Intrusion Detection Systems (IDSs). IDSs rely on known application vulnerabilities and try to detect network traffic that exploits those vulnerabilities. Therefore IDSs fail to detect traffic with unknown signature [8].

NetFlow was introduced as a solution for network management [4]. Rapidly, data exported by NetFlow started being used for many other purposes, including network planning, enterprise accounting,

Internet Service Provider (ISP) billing, network security and marketing. The System for Internet Level Knowledge (SiLK) suite uses NetFlow data for network security reasons [9]. Other tools that use NetFlow data and rely on human visual capabilities for network change detection were introduced in [10–14]. Then Machine Learning (ML) algorithms and their combination with Network Intrusion Detection Systems (NIDSs) were introduced for network traffic analysis and classification [15, 16].

Even though, there is still much to be studied. There are in the market few solutions that can show (visualise) to network administrators the impact of changes due to either newly introduced network devices or the presence of new malware in the local network as the literature review conducted for this thesis suggest.

Network failures may be anticipated if a monitoring tool could discover each new network event. Tools for automatic identification of new network events are still needed. Different authors presented different approaches to evaluate the impact of changes due to maintenance procedures. In [17, 18] flows are detected and classified if they occur within the evaluation time interval. In [19] changes are detected only on some network devices such as servers, router and switches, based on their role. The usage of machine learning to detect and classify network flows is a major research topic in network management field.

IPDiff introduces an alternative way to look to changes in traffic by identifying what is different from the previously known, based on time intervals. Since IPDiff compares flows based on time intervals then the results can be related to network maintenance operations or to malicious incidents.

1.3 Objectives

When changes occur in the network questions like the following may arise: What was the impact of the changes made on the network traffic? What went wrong? Is there traffic from this or that protocol? Is there an attack? and many more. To answer those questions, IPDiff goal is to detect network changes based on newly incoming network flows. This approach is similar to the diff Unix tool, that compares the changes made in files, i.e, the before and after state of the file [20].

The main purpose of this thesis is to develop an application, differential flow manager, which we will call hereafter by IPDiff, that will take as input network flow in SiLK format, apply some filtering and shows the differences. IPDiff only analyses security issues when a flow is considered new, thus the volume of flows to be processed for security awareness is lower when compared to the total amount of network flows.

1.4 Thesis Outline

The remaining of this thesis' chapters can be summarised as follow: Chapter 2 presents important background information about network management and its different frameworks and approaches. It also describes in some detail network flow concepts and tools used for network traffic detection and classification. Chapter 3 describes the main architectural design decisions and frameworks for web application development. It also presents IPDiff implementation and deployment environment. Chapter 4 presents the IPDiff evaluation and results discussion. Finally in Chapter 5 main findings and conclusions are presented.

Chapter 2

Background

This chapter introduces some important concepts related to network traffic change detection and classification, which are necessary to understand the main contributions of this thesis. The chapter starts with an overview on network management, protocols and frameworks. It follows by describing the application of network flows in traffic anomaly detection and classification.

2.1 Network Management

This section will introduce the network management principles, the main protocols and some well established management frameworks, starting from a centralized approach, then distributed and finishing with the web-based network management.

2.1.1 Network Management Overview

As the demand on services offered through the internet grows, ISPs are forced to expand their servers, routers, switches in quantity and processing capacity; also in their application software. This increase is also being made on the bandwidth of their communication links.

Network management is about making decisions about network performance, availability and security. Network management comprises all activities, procedures and tools to keep network equipment and services available for end users [21]. As networks management started becoming unfeasible for human manual intervention, network engineers together with software developers started finding solutions that could help to achieve that goal. Network Management Systems (NMSs) should be able to deliver services with good performance, detect changes in the network, notify administrators of failures or services outage, should detect slowness on communication links and should guarantee customers safe access to data.

Updating or upgrading network resources is a common task in networks. As networks become bigger monitoring those operations is crucial because they can be source of network performance weakness even of network unavailability [17]. The Google Gmail outage of September 2, 2009 [22] and a bug in a software upgrade in 2011 [23] are good examples to show that despite of the company, good network management practices should always be in place and undertaken by all the employees. Also computer virus in some cases can be harmful to end users and companies [24, 25]. Therefore it is important to detect and mitigate their impact over the network.

2.1.2 Network Management Protocols

As networking market was increasing more and more new network devices and protocols from different vendors and developers with different purpose were introduced. Therefore the management of network devices became much more difficult. As a consequence Internet Activities Board (IAB) recommended the Internet Engineering Task Force (IETF) to investigate a standard solution for managing equipment from different vendors. Then IETF Network Working Group has come with the Simple Network Management Protocol (SNMP) as the standard communication protocol between the network management station, a.k.a., manager and the network devices, a.k.a., agents [26]. A manager is a network station with installed software capable of managing, controlling and monitoring a network device. The agent is the device to be managed, such as hosts, servers, routers, switches and any other device SNMP capable. Agents have installed software able to collect manageable information. The information collected is defined by the Struture for Management Information (SMI) and is stored in a Management Information Base (MIB) [27]. MIB stores the objects in a collection with a hierarchical tree structure. A MIB object is defined using a subset of encoding rules of Abstract Syntax Notation (ASN.1) [28] in which each object has a unique name, a syntax and encoding. Each object type defined in a MIB is associated to SNMP operations. Thus having a MIB object, it is easy to infer which SNMP operations that can be performed. For human understanding of such object, there is a string identifying the object also called Object Identifier (OID). Enterprises have the possibility to add their own new OIDs in the structure for new object representation. To identify a manufacturer network device throughout the MIB information, the string starts from the up most top level to the manufacturer entry in the tree. As suggested by Figure 2.1, any Cisco network device should have an OID starting with 1.3.6.1.4.1.9.

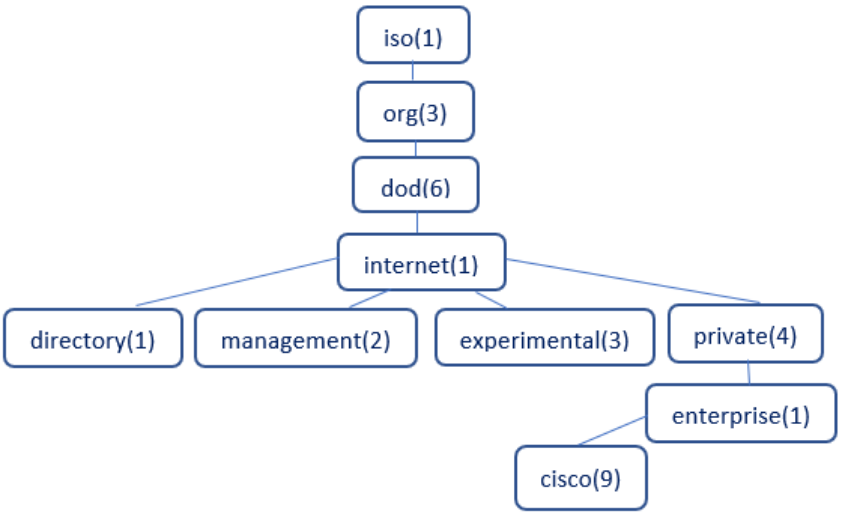


Figure 2.1: MIB Hierarchical Structure - Cisco Device

SNMP has evolved from its first version, SNMPv1, up to version 3, SMNPv3. Among others, security was the main concern behind this evolution. SNMPv1 security was based on community string authentication. Any two devices to communicate in SNMPv1 must share a community-name. This became very soon a security flaw for modification, masquerade and disclosure attacks. In SNMPv3, security was improved to guarantee data integrity, authentication, confidentiality, and others [29]. Many of the security advancement achieved in SNMPv3 started with SNMPv2 but SNMPv2 security was essentially community string based, the new features were not implemented.

SNMP is a traditional client-server protocol, thus communication between manager and agents rely on exchanging request/response messages. Managers query agents for specific information and receive

responses from the agents. Beside response message, agents also may send asynchronous messages, a.k.a. as trap, to notify the managers of the occurrence of an event (any pre-configured event that should be logged) [26]. A message may be of kind: GET, GET-NEXT, GET-BULK, RESPONSE, INFORM, SET or TRAP. Manager sends requests to agents from any available source port with destination to User Datagram Protocol (UDP) port 161 and receives the response on the same port it used for sending the request. Agents receives the request on UDP port 161 and replies (RESPONSE) using as destination port, the source port from which the packet was received. Agent may notify (INFORM or TRAP) the manager using UDP port 162, therefore the manager receives notifications and TRAP on UDP port 162. SET message are used by managers to change values of a particular variable on an agent. A trap may contain information about an alert or a notification that contains information such as a counter of IP datagrams discarded due to an error on the IP datagram header, or the version of the software running on the DNS server, or the status information of the device interface.

Currently, any network device with Transmission Control Protocol/Internet Protocol (TCP/IP) implementation to be managed must support SNMP, thus the device should adopt SMI, MIB and SNMP for its management [27].

There is also the Remote Monitoring Protocol (RMON), which is primarily used for remotely monitoring networks. RMON was introduced as a solution to minimize the impact of management information traffic on network resources, namely, bandwidth availability or on failure to communicate with the manager. For that RMON uses stand-alone devices, also known as monitors or probes, devoted solely to network monitoring. It defines objects for monitoring remote network devices shared between RMON agents and the monitor [30]. RMON agents uses SNMP to continuously query for local traffic statistics even when there is no communication with the monitor. Whenever needed the monitor request for available monitoring information from the agent. RMON agents are capable of checking access control over information. When exchanging information the message are encrypted. RMON security rely on SMNP security.

Syslog is used to log notifications on operating system events [31]. The Syslog protocol uses a layered approach for separation between message content and its transportation. The protocol uses hierarchical structured data elements to store the information. The information is collected by the originator device and is transmitted to the collector device. Syslog may be attacked in mainly two phases [32]. In the first phase an attacker may intercept the information while it is being transmitted from the originator to the collector. Another attack profile occurs when an attacker tries to get access to log file to read, modify or even delete its content. Syslog uses end-to-end encryption to keep confidentiality on transmitted data.

There are other widely used application layer protocols that help network managers to perform their tasks. One example is telnet, used for remotely access a network device and manage the network from it. Many network devices have embedded web server and can be managed trough the Hypertext Transfer Protocol (HTTP), which is the standard protocol for accessing devices through a web interface. Due to security flaws, both protocols were replaced by others more secure. The Secure Shell Protocol (SSH) [33, 34] protocol is used to remotely access network devices in a more secure way, by creating a secure communication tunnel on which transmitted messages are encrypted using asymmetric encryption. Hypertext Transfer Protocol Secure (HTTPS) [35] extends HTTP by addition of Transport Layer Security (TLS) on data exchange. It replaces HTTP on much web-based network monitoring software [36].

2.1.3 Network Management Frameworks

Network management standardization process started when the International Telecommunication Union -Telecommunication Standardization Sector (ITU-T) introduced the classification of network management task into five functional groups as: Fault, Configuration, Accounting, Performance and Security (FCAPS) [37, 38]. The description of each group is as follow:

Fault Management involves all the preventive and corrective measures to keep network devices available.

Configuration Management defines the installation, configuration and upgrading of network devices procedures. Defines a Network Operation Center (NOC), a central point where network is controlled and monitored.

Accounting Management provide the usage information for accounting and billing purpose.

Performance Management deals with all network devices statistical information for performance analysis.

Security Management control access permissions to network devices, provide to authorized users a safe access to network devices and its data.

The ITU-T standardizing process was too slow when compared to the rapid growth in new devices, software and services, thus the network operator and protocols developers have recommended changes in the standardization procedures [39]. Other researchers started proposing that the NMS should evolve to replace human interpretation of network events. The changes also pointed towards the convergence of network protocols from different vendors. Therefore new NMSs should provide common interfaces for internetworking. As a result new management models were proposed [40].

2.1.3.1 Distributed Network Management

In the SNMP architecture, agents do simple tasks for reporting their status while the managers perform most of the tasks. As network size started increasing it became clear that this model could not scale. The first approach toward decentralized managers was introduced with SNMPv2, that introduced the concept of proxy to deal with compatibility problems due to the coexistence of both SNMPv1 and SNMPv2 [41]. In this case, the manager would exchange information with the proxy then the proxy would interact with the agent. RMON was the next step towards Distributed Network Management (DNM) with the introduction of the concept of probe or monitor. Monitors took care only of monitoring a set of collectors (agents) and then could report to the manager.

The main purpose of Distributed Object Computing (DOC) is to provide distributed applications for network management. With the Object Oriented (OO) paradigm the separation of services and applications can be accomplished within certain criteria. Also, the OO paradigm with its concept of interface helps to set some generalization interface for communication between different protocols.

Several discussion can be found in [42] about distributed management systems such as Distributed Big Brother (DBB), Distributed Management Environment (DME), Hierarchical Network Management, Management By Delegation (MbD) and Common Object Request Broker Architecture (CORBA). Between those proposals, the CORBA framework has been widely adopted solution for DNM [42–44]. Since CORBA has a good support services, DOC has been used for managing heterogeneous network domains also it was suggested as the replacement platform for the traditional SNMP based NMS [38,45]. The CORBA framework has been proposed for interoperability between the two main management

frameworks, that is, SNMP for TCP/IP networks and the Common Management Information Protocol (CMIP) for Telecommunication Networks [38, 46].

2.1.3.2 Web-based Network Management

SNMP based network management uses two communication schemas for data acquisition In the first a request/response model is used in which a manager polls data from agents. In the second schema, managers receive unsolicited notification, thus a push data model is used.

A publisher/subscriber model was presented in [47] as the first attempt toward a web based network management framework. In this model data acquisition takes tree steps. In the first step agents announces which MIBs and SNMP notifications they can send. In the second step each managers subscribes to the MIB they are interested in. And in the last step, at any scheduled time, agents may send the data to a manager. This approach reduces the amount of management traffic in the communication links.

In addition, web based network management framework also solve the problems of deployment time and the interoperability between existing solutions from different vendors. The web framework comprises protocols and data model. This framework introduced in the network management field the concept of web gateway which acts as an adapter between web (HTTP request) and the SNMP or CMIP environments. An example of such frameworks is the Web-Base Enterprise Management (WBEM) [38, 48].

2.1.4 Network Monitoring Tools

In the field of network management, a network monitor is a system, usually a software, that observes network devices and notifies the network administrators when pre-defined conditions occurs, by sending an e-mail or by generating an alarm that will be visible to the network administrators. The condition is, for instance, a failure of some kind.

The Internet Control Message Protocol (ICMP) messages are used in many network monitoring tools. ICMP defines sets of messages and policies used by hosts to communicate their status, mainly for routing purpose or for error reporting [49–51]. An ICMP message is contained within the IP packet and its header has 8 bytes with the first byte reserved for the Type, the second byte for the Code, followed by two bytes for the Checksum. The remaining 4 bytes depends on the type and code of the ICMP message. Each ICMP message has its own datagram format and extensions have been made to support new protocols. Table 2.1 shows some of the ICMP messages used by the ping and traceroute commands.

Table 2.1: ICMP Message Type

Type	Message
0	Echo Reply
3	Destination Unreachable
8	Echo Request
11	Time Exceeded

The Packet Internet Groper (ping) is one of the most primitive tool used for network monitoring. A ping command uses several ICMP messages as shown in Table 2.1. The echo request message (Type 8) is sent to its destination host. Any host receiving this type of message must reply with its status, otherwise the destination host is assumed to be offline. If the destination host is available, it replies to that message with an echo reply (Type 0), as in Figure 2.2. However the destination host may reply with

any other type of message, e.g., for security reason, a host may be set to reply only to requests from known source, in this case, the destination host could just reply with destination unreachable message (Type 11). It is important to say that ping can be used for attacks like Denial of Service (DoS), with

```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. Todos os direitos reservados.

C:\Users\ACERPC>ping 41.221.245.42

Pinging 41.221.245.42 with 32 bytes of data:
Reply from 41.221.245.42: bytes=32 time=264ms TTL=39
Reply from 41.221.245.42: bytes=32 time=264ms TTL=39
Reply from 41.221.245.42: bytes=32 time=265ms TTL=39
Reply from 41.221.245.42: bytes=32 time=265ms TTL=39

Ping statistics for 41.221.245.42:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 264ms, Maximum = 265ms, Average = 264ms

C:\Users\ACERPC>
```

Figure 2.2: Ping Command Output

ping-of-death [52] being the most famous of its kind.

Another command used for network monitoring is traceroute (tracert for Windows Operating Systems). Traceroute uses two ICMP messages with increment by 1 of its Time-to-live (TTL) value. The process starts when the source sends a probe packet towards the destination with an initial TTL value of 1. The process end when the destination receives the packet and replies with ICMP messages Destination Unreachable (Type 3). Each router, in the path may forward the packet when its TTL value is still greater than 1 upon decrementing it. Otherwise the router will reply with the ICMP messages Time Exceeded (Type 11), if its TTL value hits 0 upon decrementing it [5]. Figure 2.3 shows the use of tracert command. Traceroute is also used combined with Border Gateway Protocol (BGP), a network layer

```
C:\Users\ACERPC>tracert 41.221.245.42

Tracing route to mail.unimetro.ed.ao [41.221.245.42]
over a maximum of 30 hops:

  1    <1 ms    <1 ms    <1 ms  192.168.1.253
  2    <1 ms    <1 ms    <1 ms  gatekeeper1.ist.utl.pt [193.136.134.157]
  3    <1 ms    <1 ms    <1 ms  brites1.utl.pt [193.136.134.10]
  4     1 ms     1 ms     1 ms  gt3.fccn.pt [193.136.1.93]
  5     1 ms     1 ms     1 ms  ROUTER4.10GE.CR1.Lisboa.fccn.pt [193.137.0.4]
  6     1 ms     1 ms     1 ms  fccn.mx2.lis.pt.geant.net [62.40.124.97]
  7    16 ms    12 ms    12 ms  ae0.mx1.mad.es.geant.net [62.40.98.107]
  8    32 ms    32 ms    32 ms  ae7.mx1.gen.ch.geant.net [62.40.98.67]
  9    40 ms    40 ms    40 ms  ae1.mx1.fra.de.geant.net [62.40.98.109]
```

Figure 2.3: Tracert (traceroute) Command Output

protocol for routing packets intra and between Autonomous System (AS) [53].

Another tool used are log files. Log files may hold host state, its operating system information, local disk usage information and network access. They may hold information of a specific service, such as mail server log. A good security log should be descriptive, relatable and complete [4]. To get profit of information stored in log files many tools were developed, such as logcheck that inspect log files to find security anomaly based on patterns stored in a database [6]. The main problem with log files is finding meaningful information on it. A tool that uses clustering algorithms for mining and classifying the log files data has been introduced in [54]. A similar tool has been proposed in [55] but considering an interactive process executed every day.

2.2 IP Network Traffic Flow

In this section a brief introduction to some key concepts on network flow and the protocol for its management are described.

2.2.1 Network Flows

Flow-based measurements have been widely used for network management. Those measurements may be collected by any network device configured to monitor the traffic passing through its network interfaces. Such devices are known as sensors [4]. The location of a sensor in the network has impact on collected data. Also a sensor depends on the type of measurements to be collected. In practice a sensor may be a switch, a router, firewall, a tap or a software. It is a challenge to build sensors since that a good sensor should be complete, that means being able to describe any single network event, and should also be non redundant in a sense that should not replicate information about events. There exists essentially two groups of sensors: network-based and host-based sensors. They can be implemented as physical devices or as software. In most cases, sensors when physical are located on core network switches or routers. In case of software sensors they are almost installed and configured on network servers. Network-based sensors are the most used option in security awareness analysis because host based analysis tools may fail to detect previously unknown attacks or zero-day-attack.

2.2.2 IP Network Flow Protocols

NetFlow was developed by Cisco, initially to solve switching problems [56, 57]. A flow is a collection of packets with identical Internet Protocol (IP) attributes such as source IP address, source transport port, destination IP address, destination transport port, and transport protocol (TCP/UDP), Type of Service byte, input interface index, that passes through a probe within a specific time interval [4,58,59]. NetFlow records are kept in cache for at least 15 minutes when inactive and up to 30 minute when active (long lived flow). These default values can be changed via software configuration parameters. From time to time the content of cache is exported to a NetFlow exporter using UDP packets. The protocol used to export the NetFlow cache is known as NetFlow Protocol. The flows in the cache are grouped based on similarity of its parameters. That is, the first packet arriving would be used to create an entry in the NetFlow cache. When a second packet arrives its parameters are compared trying to find a match If a similar flow already exists, the number of packets and bytes are updated accordingly to the new values. If no similarity found, a new flow is created as shown in Figure 2.4.

NetFlow became widely used for network monitoring and traffic analysis. In the meantime several other versions from different vendors appeared in the market, such as jFlow from Juniper [60], NetStream [61] from Huawei and CFlowd from Alcatel. As could be expected interoperability problems arose between network equipments from those vendors. NetFlow has evolved from its version 1 up to 9 (NetFlow version 9) [57], as for the date of production of this work. NetFlow version 5 and NetFlow version 9 are the version most deployed on enterprise networks.

Another technology used for monitoring traffic in data networks is the sFlow. For monitoring network devices such as routers and switches, sFlow defines sampling rules, sFlow MIB and the format of sampled data. An sFlow agents creates and forwards data to a central flow collector or sFlow Analyser. sFlow was designed to addresses some issues that occurs when monitoring network traffic at Gigabit speeds and higher with low cost agents [62]. However, sFlow has been considered as packet sampling [63].

Due to the many different formats that different vendors use for flows, IETF has introduced the IP Flow

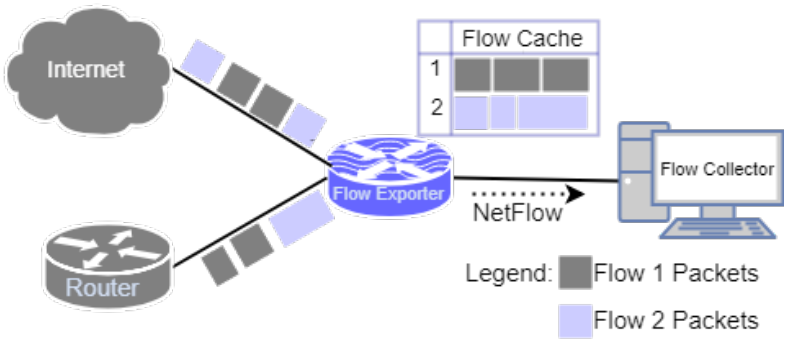


Figure 2.4: NetFlow Overview

Information Export Protocol (IPFIX) [59] as the standard protocol for flow export. The IPFIX was based on NetFlow version 9 from Cisco. The IPFIX export format uses a template which provides enterprise specific extension field. Therefore an enterprise may use the extension field to specify additional private flow information.

2.2.3 Flow Collectors

Flow collectors are responsible for processing and storing the flow information received from sensors. There are several flow collectors in the market [63,64]. The NetFlow Probe (nProbe) has been presented as a complete solution for network flow acting as a probe (sensor) and a collector. It can be deployed as an embedded solution and provides a web interface to overcome usability problems introduced by others command line based collectors, e.g, ntop [65]. The Yet Another Flowmeter (YAF) [66] is a flow collector software based on libpcap and can read pcap data from files or capture packets directly from a network interface. YAF can expands NetFlow to add new features used by the SiLK suite. It exports flow data in the IPFIX format over Stream Control Transmission Protocol (SCTP), Transmission Control Protocol (TCP) or UDP transport layer protocols. It supports Berkeley Packet Filter (BPF) for filtering incoming traffic. Another interesting feature is that YAF can export encrypted flows using TLS. Yaf commands supports several options [67]. The options available are:

--**live** specifies the type of data being read, e.g, pcap data.

--**filter** applies a BPF filter to the pcap data.

--**out** to specify the output file.

--**silk** specifies the output that can be parsed by SiLK tools.

nfdump is another flow collector which collects and stores data in a binary format. Due to his file format nfdump can handle more information if compared with YAF [64]

2.3 Network Traffic Visualization

Human visual capabilities are explored in network management thus this section describes some of visualization solutions that has been used for network management. The section introduces first packet based visualization tools then the flow based visualization tools.

2.3.1 Packet Visualization Tools

Today there are lots of visualization tools. For the purpose of the study and space limitation we present here those that we found to be more relevant in the filed. MRTG [7] is a visualization tool that uses SNMP to collect visualization information and renders an Hypertext Markup Language (HTML) page to display that information. The capability of presenting time-series information up to 1 year was one of its main achievement. For that, MRTG initial version used to keep records of increasing log files which was its main disadvantage. To overcome the problem of increasing log files, in MRTGv3 data is saved in Round Robin Database (RRD) of log files. RRD is capable of limiting the size of log files by storing data in a circular based database. RRD instead of directly storing the data it receives as input, firstly it sample them and then it stores the resulting data, which can be considered to be a new version of the original data.

The main limitation of RRDTool is that it is only capable of dealing with time-series data. Paessler Router Traffic Grapher (PRTG) [68] offers several visualizations features which makes it a powerful network monitoring tool. Besides SNMP, it also uses NetFlow and WMI (Windows Management Instrumentation), sFlow and jFlow as a source of its data. Its alerting and reporting capabilities are used for network forensic analysis.

2.3.2 Flow Visualization Tools

FloVis was defined as a suite of iterative visualizations tools, each one displaying a specific set of network flows [10]. It uses as flow data source the SiLK suite. For flow processing SiLK data are converted in a relational database format. FloVis has three main visualization modes: (1) Activity Diagrams that monitor hosts network activities and visualize them in one of the time-series colour codes. (2) Connection Bundles that monitor and visualize the interaction between hosts and (3) the NetBytes that perform inspection of a particular flow upon analyst request.

The Security Incident Fusion Tools (SIFT) is capable of gathering, filtering and analysing network flows and visualize the results in a single window [11]. The suite is composed by two visualization tools, namely NVisionIP [69] and VisFlowConnect-IP [70], that can be used separately. SIFT suite uses CANINE [71] for flows conversion from different network sensors in a single format. SIFT flow format is known as NCSA format. Thus SIFT can be used for NetFlow visualization independently of the source of the NetFlow data.

Both visualization tools have different concerns. NVisionIP deals with loading NetFlow data from files or network sensors and allowing forensic queries on them while VisFlowConnect-IP is concerned with interaction hosts within local network or to the internet. The suite rely on visual diagnostic of traffic anomalies. It allows to drill down a specific area of the screen being displayed for more detailed information.

VIAssist [12] is another network visualization tool. It is aimed to link network flows using multiple views. It presents different levels of detail on each view. The tool uses the SiLK suite as flow input and provides a customizable dashboard to present an overview of network flows and allows automatic schedule of flow analysis. Network flows can be tagged and annotated to allow sharing network security data among the analysts. VIAssist uses Isis interface that allow showing a high level overview of data and drill down into them for more detailed inspection if needed. To solve the scalability issues imposed by reading data from files and using relational databases, VIAssist uses compressed SiLK data files and a relational database.

FlowScan is a suite of open source tools [13]. It uses the cflowdmux and cflowd programs, which are

components of cflowd[1]. cflowdmux collects NetFlowv5 records which are forwarded to cflowd that process them and produces as output flows formatted according to FlowScan. The other component is the Flowscan, a perl script, which is the central process that loads the FlowScan modules according to the analysis to perform. FlowScan also uses RRDTool for storing time-series data that can be aggregated into averages of time intervals. This provide to FlowScan diversity and reacher visualizations, such as, flow-per-second, packet-per-second and byte-per-second graphs.

Nfsight was presented in [14] as an application designed to overcome most of the issues not solved by others tools, like FloVis and SIFT suite. Nfsight is made of three components: (1) Service Detector that analyses unidirectional NetFlow flows to identify clients and server assets. (2) Intrusion Detector that detects suspicious activities through a set of graphlet-based signatures.(3) A front-end visualizer that allows the administrators to query, filter, and visualize network activity.

The Nfsight Service Detector uses the concept of End-point when identifying network elements. It considers as end point the tuple IP address, IP Protocol, Port number. An end point may represent a client or a server. End point identification are based on some heuristics, such as: (1) Flow timing: assumes that in a bidirectional flow the server should be the one with more recent timestamp and the client otherwise. (2) Port Number: assumes the server to be the end point with smaller port number.

2.4 Change Detection in Network Traffic

For network traffic change detection it is important for network administrator to be able to identify what is a new traffic flow. Many approaches are used to achieve this goal. In the sections below detailed information is shown.

2.4.1 Changes On Estimated Traffic

The concept of scketch is applied to any simplified object that can uniquely identify a more complex object or concept. Its main purpose is the reduction of memory space and computation. The concept of scketch can be implemented in very different ways depending on the purpose, for instance, by using an array. In such case, the performance depends on how the array is indexed and what values are stored in. An example of indexing the array that stores network flow data, could be a time period or a source IP address [72, 73].

A method that detects changes based on sketches was introduced first in [74]. In that work, they use the concept of k-ary scketch to store network traffic flow summaries and to detect the changes by comparing them with previous knowledge and to produce forecast errors for each flow. Then significant errors are used to spot new traffic. A similar study, that uses the sketch approach was presented in [75]. A new concept of deltoid was introduced to measure relative or absolute differences on network traffic over time or between routers and their interfaces. They proposed a tool capable of monitoring, filtering and grouping flows based on their needs, e.g. number of distinct flows, number of source or destination IP addresses or even the number of TCP connections. At the end, they compute the differences from the estimated metrics.

A slightly different method that uses the concept of prediction on traffic matrix behaviour was presented in [76]. Each matrix entry holds flow informations based on time interval, so the size of the matrix may vary over the time. To compute the new traffic they compare the flows in the matrix with the estimated traffic.

[1] cflowd is no longer supported by CAIDA as its flow collector - http://www.caida.org/tools/measurement/cflowd/

2.4.2 Changes Due to Maintenance

Mercury [17] was presented as a tool capable of detecting behavioural changes that persist after a maintenance operation by applying a ranking on changes. PRISM was presented by [18] as software for network performance changes detection. PRISM produces alarms on performance behaviour change that includes sudden increase or decrease in a performance metrics after the maintenance operation occurs. The detection of the changes depends on the day and time the service measurements (CPU and memory utilization, packet counts, packet losses) are collected. To minimize the effect of the time on collection measures PRISM uses SNMP MIBs and device syslog. Although PRISM could be tested and had shown a good correlation between maintenance interval and behavioural changes it is not absolutely correct to assume that the changes were caused by the maintenance.

The same group of researchers have then presented a new tool to cope with PRISM's time dependancy. They introduce FUNNEL [77] that could detect the impact of maintenance changes on the key performance indicators (KPI) immediately

SoNSTAR [78] was presented as a solution that helps network administrator to do in parallel multiple tasks, including monitoring the network. SoNSTAR rises a sound when a particular change occurs. For that they map flow events to sounds so that a particular sound is triggered by the event

2.4.3 Statistical Change Detection

Another work on change detection introduced a concept of equilibrium and a tool named A Short-Timescale Uncorrelated-Traffic Equilibrium (ASTUTE) [79]. The tool defines two main variables, one that holds the evaluation timescale and another that holds alert threshold used to control the false positive rate. Upon flow arrival, their start time and duration are used to define active flows. Active flows are stored in a vector which is then processed. When the threshold is aged ASTUTE tries to classify that anomaly. Since the classification process is only made on threshold aged flows, ASTUTE is considered by its authors to be computationally simple. The ASTUTE experimental results has shown that ASTUTE may detect new anomalies even without a training phase.

Lakhina et al. have shown a new method for diagnosing behavioural statistics changes in network based on the separation of the traffic in two subspaces with normal and anomalous traffic from each pair origin-destination (OD) IP address [80].

2.5 Network Traffic Analysis and Classification

The ease of development and publication of new applications running on the Internet has increased the diversity of traffic entering and leaving a network. This phenomenon brings great challenges for network administrators and particularly for data security. With this many tools and researches related to this field have been proposed. In following subsections different methodologies and some tools are proposed for the classifications.

2.5.1 Classification Methods

Traffic classification can be undertaken by TCP or UDP source and destination ports, deep packet inspection, packet flows (statistical) or lately by using some clustering algorithms.

Traditionally network traffic classification was based on the well known TCP or UDP ports. This is the oldest approach and it is based on classifying the traffic arriving to a sensor's interface based on source and destination TCP or UDP well-known port or user assigned ports [3,81]. It was a very successful way

of classifying network traffic because most applications were configured to use their default ports. This method became inappropriate to achieve network traffic classification with the arrival of new application that for security reasons started using different assigned or not assigned ports.

Packet inspection was the following approach used for traffic classification. This approach classifies the network traffic by inspecting the content of a packet arriving to the host network interface. This can be seen as signature based, since that its results are based on finding patterns in the payload. Several studies have explored signature-based methods [82–84] and software such as Snort [85] detects anomalies based on known signatures. However the approach fails in a presence of a new behaviour or encrypted data [74, 86].

With the development of NetFlow and its frequent use for traffic analysis, several studies began to address the effectiveness of using NetFlow for traffic classification. This approach bases its classification criteria on flow statistics, e.g., it can classify traffic based on IP address, protocol, packets and behavioural information.

Comparisons among those criteria have been made in [87] and found that port based classification still have its relevance in presence of legacy applications. For their classification they have used some metrics such as precision, recall and aggregate precision. The metrics were defined as: (1) Precision is the ratio of True Positives (the number of correctly classified flows) over the sum of True Positives and False Positives (the number of flows falsely ascribed to a given application). (2) Recall is the ratio of True Positives over the sum of True Positives and False Negatives (the number of flows from a given application that are falsely labelled as another application). (3) Aggregate precision is the ratio of the sum of all True Positives to the sum of all the True Positives and False Positives for all classes. However the study also shows that when the same metrics are applied, port based approach fails in presence of applications that uses ephemeral ports or when an application uses a port that is masqueraded in another application or in case of port overlapping between two applications in the same network. An interesting outcome of this study suggests that traffic classification using a single approach may not be enough to capture all network traffic. However another study presented in [88] shows that NetFlow can be used to classify most of network traffic including P2P applications such as file-sharing (BitTorrent and eDonkey) and Skype, with high rate of accuracy.

Other Netflow based classification study was undertaken in [89]. In that study the ML algorithm C4.5 was used to avoid human intervention on classification process [15]. The decision tree C4.5 algorithm was used as classifier, to label the unknown NetFlow data from the dataset. By applying sampling rate to those NetFlow data, the study has concluded that the rate of accuracy in the application classification was lower compared to the achieved when using not sampled NetFlow data.

A combination of statistical flow and the machine learning techniques to classify applications was made in [90]. For known flows a supervised ML algorithms was used for flow classification while for unknown flows ML clustering techniques were applied for flow labelling. At the end, a traffic classifier was build.

2.5.2 Traffic Analysis

Traffic analysis is used in this paper for the process of inspecting a packet in order to find out patterns that can be used to build security information knowledge. In most cases it will be used to find out protocols in use or for searching patterns in the content of the packet or packets being analysed.

Protocol analysis is used to monitor network traffic. It can be seen as a probe. It identifies the information contained in a packet and tries to understand their structure and relationship to gather security evidence [91, 92].

TCPDump is a command line tool for intercepting and analysing packets traversing the network in-

terface being monitored. Its report capabilities allows it to print packets timestamp, protocol, source and destination IP address and ports, and much more. It is based on libpcap library on Unix like operating systems. For Windows operating system TCPDump appears as WinDump which uses the Winpcap library for the same purpose. TCPDump has much more applicability [91].

Wireshark is one of the most popular and powerful packet capture and analyser in the market [93]. Wireshark comes with one executable that has a graphical display and many others being Dumpcap and implementation of lipcap used for live packet capture. Its graphical interface has filtering and dessication features for almost all network packets [94]. Thus its widely used from network monitoring and intrusion detection purpose but also for hacking systems [95].

2.5.3 Flow Analysis Tools

Packet analysis tools are essentially host-based. Host-based solution are not capable of identifying new attacks or anomalies. Flow analysis has become the most used approach for network forensic analysis because flow analysis examines sequences of related packets [92]. In the rest of this section we present the tools and techniques used in the field of flow analysis.

Audit Record Generation and Utilization System (Argus) [96] is a libpcap-based network flow sensing, collection and analysis toolkit. The toolkit has two main components, a Argus-server that reads packets from network or from files. Its function is of a network sensor that can output flow export data in Argus' compressed format. The other component is the Argus-client tools that is used to collect, distribute, process and analyse Argus data [97]. The tool has filtering capabilities that allow analysts to customize the data being analysed. It also supports multiple output streams, which can be directed to different flow collectors.

The SiLK, is a collection of software tools developed by the Computer Emergency Readiness Team (CERT) for Network Situational Awareness (NetSA) (CERT/NetSA) to facilitate NetFlow security analysis in large-scale networks [1, 4, 9, 98].

In this section the SiLK suite is presented with much more details when compared to Argus since that it will be used as the NetFlow collection tool for this work. On the date of writing this document SiLK was only available for Unix and Linux platforms.

The YAF SiLK flow collector collects NetFlow v5, NetFlow v9 and IPFIX flows. When compiled with the *libfixbuf* support, SiLK also support sFlows. Those flows are converted into a SiLK flow format, which is more space efficient, then stored into binary flat files that can only be read with SiLK commands. SiLK directories and files are organized according to the time of conversion process [99].

SiLK is considered as a database at command line because it is composed of several tools each of them performing specific tasks such as querying, manipulation or aggregation of data [4]. Another very useful feature of SiLK is that commands can be chained along pipes to produce a unique complex but powerful command according to the needs. As for the flow record source, SiLK uses YAF as its standard flow collector. Other sources for collection flow data are the *rwptoflow* and *rwtuc* which convert flows in different format into SiLK compressed format which is a compact binary format. Thus can not be read directly but with the rwcut tool. The *rwcut* command can print up to 29 different fields [4]. The *rwcut* as other SiLK commands' uses the *--fields* option to list which fields to be printed and this can be achieved either by specifying the fields' name or their numeric value. Actually there are much more fields than those shown in Table 2.2. Other *rwcut* options are *--num-recs* used for limiting the number of output records, and *--icmp-type-and-code* when used places ICMP type in the source port field and the ICMP code in the destination port field of the flow record.

For most of the forensic analysis filtering fields is used. For SiLK this is done by using the command *rwfilter* which also has many options for filtering purposes. It reads input from a file or through a pipe

Table 2.2: SiLK fields of interest

Field	Numeric ID	Description
sIP	1	Source IP address
dIP	2	Destination IP Address
sPort	3	Source port
dPort	4	Destination port
protocol	5	Layer 3 protocol
packets	6	Packets in the flow
bytes	7	Bytes in the flow
flags	8	OR of TCP flags
sTime	9	Start time in seconds
eTime	10	End time in seconds
sensor	12	Sensor ID
type	21	Type of the flow

and applies the filtering options specified with the command. For filtering purposes the *--pass* option is used. If a flow matches the filter's rules it passes, if not it fails. For the purpose of this study the *rwfilter* options *--stime* and *--etime* which filters based on time window are the most interesting. Filtering can also be done on source or destination IP address, source or destination ports, on protocol, on TCP flags, on start and end date of the flow and much more other options are available. The SiLK command textitrwfilter combined with *rwcut* command is of the most valuable for analytical purposes.

```
[root@localhost SiLK-LBNL-05]# rwfilter --daddress=131.243.104.x inweb/2005/01/06/iw-S0_20050106.20 --pass=stdout | rwcut --f
ield=1-9,21 --num-recs=9
            sIP|            dIP|sPort|dPort|pro|   packets|     bytes|   flags|                 sTime|   type|
 150.52.105.212|131.243.104.181|   80| 1262|  6|         1|        40|F   A   |2005/01/06T20:22:49.477|  inweb|
 150.52.105.212|131.243.104.181|   80| 1263|  6|         1|        40|F   A   |2005/01/06T20:22:48.920|  inweb|
  59.100.39.174| 131.243.104.27|   80| 3188|  6|         3|       128|FS  A   |2005/01/06T20:22:57.173|  inweb|
  59.100.39.174| 131.243.104.27|   80| 3191|  6|         3|       128|FS  A   |2005/01/06T20:22:59.964|  inweb|
  59.100.39.174| 131.243.104.27|   80| 3193|  6|         3|       128|FS  A   |2005/01/06T20:23:04.820|  inweb|
  59.100.39.174| 131.243.104.27|   80| 3189|  6|        14|     14812|FS PA   |2005/01/06T20:22:57.208|  inweb|
 150.52.105.212|131.243.104.181|   80| 1264|  6|         6|       578|FS PA   |2005/01/06T20:22:55.028|  inweb|
 150.52.105.212|131.243.104.181|   80| 1267|  6|         4|      3088| S  A   |2005/01/06T20:23:05.097|  inweb|
 150.52.105.212|131.243.104.181|   80| 1268|  6|         4|      3088| S  A   |2005/01/06T20:23:05.098|  inweb|
[root@localhost SiLK-LBNL-05]#
```

Figure 2.5: SiLK rwfilter command output

When combining both commands Unix pipes are used to redirect the standard output. Thus the rwfilter output becomes the input data for rwcut as can be seen in Figure 2.5

To create time-series data the SiLK suite provides the *rwcount* command. SiLK provides also other commands such as *rwstats* used for statistical analysis, the *rwuniq* that can be used alone for counting uniq flows or combined with *rwstats* for comparisons. The other and most powerful commands and claimed to differentiate SiLK from the other similar flow tools [4] are *rwset*, *rwsetbuild* and *rwsetcat* which works with IP sets in textual or in binary format. IP set is defined as a binary representation of a collection of IP addresses. SiLK can create those sets from text files or from SiLK data. rwset command when combined with the $--sip\text{-}file$ or $--dip\text{-}file$ will produce files containing source and destination IP addresses from the flow records. Actually there are lots of commands that can be found from the SiLK documentation [100].

For usability purposes of the suite by the security analysts, iSiLK [100] was developed. It is a graphical front-end for the whole SiLK suite and uses the SSH protocol to connect to an analysis server.

2.5.4 Flow-Based Intrusion Detection and Analysis

The traditional approach used for intrusion system were based on deep packet inspection to find out known patterns. This approach is not applicable for current high speed networks for performance rea-

sons. At the other hand packet inspection tools fails when in presence of unknown (new) attacks.

Thus much research have pointed out their study toward flows as the new source for intrusion detection and prevention analysis. Sperotto et al. have presented [16] some attacks such as DoS, SYN Flooding, scans and others, that are detectable when using flow records. They have pointed out that the absence of payload on flow records has an impact on intrusion detection systems. Thus a well designed security system should include both deep packet inspection techniques to be activated upon flow anomalies are detected by flow based intrusion detection system.

Abdalla et al. in their study [101] have used a combination of flow records captured snort alerts with machine learning algorithms for traffic classification and have achieved around 75% of True Positives and 0% of False Negative on detecting flows containing malware in payload. A similar comparison made in [102], has shown that tuning besides reducing the datasets, it maintain the detection accuracy at acceptable levels.

An approach that combines ML and NIDS to detect malicious traffic in network flows has been presented in [103]. The study has introduced a tool called FlowHacker which identify malicious traffic without requiring previous knowledge.

2.6 Summary

This chapter has described some important concepts about network management and have shown several research and solutions developed towards an effective NMSs. It has also been shown, how network flow can be used to answer the questions that arise, when there are changes on the network traffic.

The literature review presented in this chapter has highlighted several methods for detecting changes and classifying network traffic. The use of ML algorithms combined with IDSs has be shown to be the trend for network anomaly detection and traffic classification. However for the development of IPDiff the approach used in [79] was considered the most appropriate to follow but without the need of defining an alert threshold.

Chapter 3

IPDiff Design and Implementation

This chapter provides an overview of the architecture of IPDiff and describes its implementation. The chapter starts by showing the IPDiff description and requirements. Secondly the chapter shows the IPDiff modules and the associations between them. In addition, the frameworks used are briefly discussed. In sequence, the chapter presents the algorithms used to process the flows. Finally, it ends with the description about IPDiff implementation and deployment environment.

3.1 IPDiff Architectural Design

This section describes the design decisions taken when developing IPDiff.

3.1.1 IPDiff Description

IPDiff receives two input parameters which are:

- A source file which contains the flows to be analysed
- A time interval used for flow comparison

Based on the time interval, IPDiff classifies flows in four categories as:

1. Old flows, those that happened before the interval
2. New flows, the flows that were active during the time interval
3. Missing flows, the flows that existed before the beginning of the interval but not after.
4. Diff flows, are those flows that were classified as new flows and never happened before, same is saying that any diff flow should appear as new flow but should not be present in old flows.

We aim IPDiff to produce useful information on top of diff flows and missing flows, e.g., country origin of the source and destination IP addresses, BGP Autonomous System Number (ASN) for the given IP addresses, protocol and source ports used. Moreover, we aim IPDiff to produce security awareness information based on known reputation of the ASN found in diff flows. Those outputs should help evaluating the current status of the network. The Figure 3.1 shows the IPDiff system context diagram.

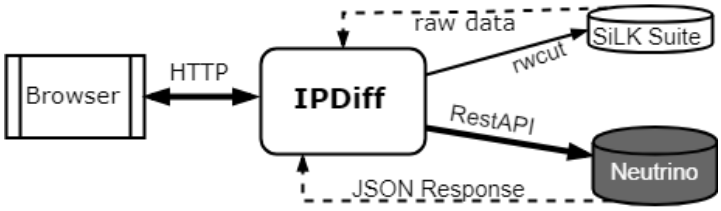


Figure 3.1: IPDiff Context Diagram

3.1.2 Requirements

A requirement can be a function or a service that the system can do and its operational environment constraints [104]. They are divided in two groups: (1) functional requirements state what the system should and should not do and its behaviour under some particular inputs or situations when interacting with its environment [105]. (2) Non-functional requirements are the system restrictions that may have impact on the design decisions phase, points out a deployment infrastructure, set minimum system performance or efficiency metrics, may be a constraint on the time to the market, etc. [104].

Based on above definitions, the IPDiff requirements are as follow:

- Take as input the SiLK flow files.
- Compare flows based on time interval
- Classify the flows in old flows, new flows, missing flows and diff flows.
- Group flows by country, protocol, source IP address and AS number.

3.1.3 IPDiff Modules

In software architecture, the concepts of architectural views are widely applied when designing any software. A view represents a set of system elements and associations between them [106, 107].

In object oriented programming, system elements can be represented as objects that behave as those elements [108–110]. According to the list of requirements presented in Section 3.1.2, IPDiff will be implemented as a web application, thus a client−server architectural model is assumed. For showing the responsibility of each system element and the relationship between them, the module decomposition viewtype was adopted [104, 106]. The IPDiff main components are shown in Figure 3.2. From any web browser, an HTTP request made to the IPDiff web page will be handled by the Glassfish application server. Then the HTTP request input parameters are forwarded to the IPDiffBean module which in turn uses the Reader to create flows from the data read from the submitted file.

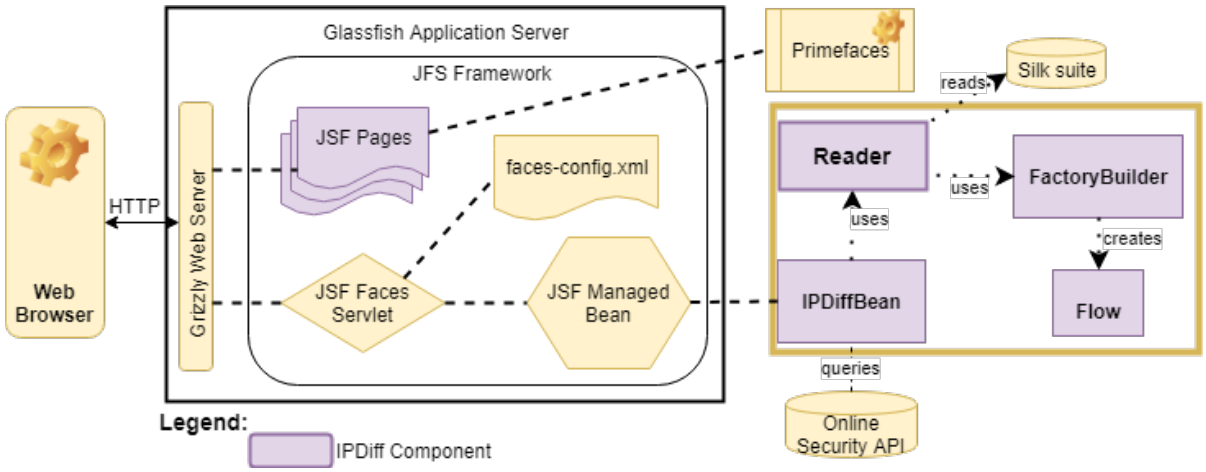


Figure 3.2: IPDiff Decomposition Style

3.1.4 Class Diagram

Structural models show the organization and architecture of a system. Unified Modeling Language (UML) class diagram are widely accepted and used among software designers to describe the software structural components and their associations [104, 111–113]. The IPDiff class diagram can be seen in Figure 3.3. A short description of the IPDiff main classes comes below:

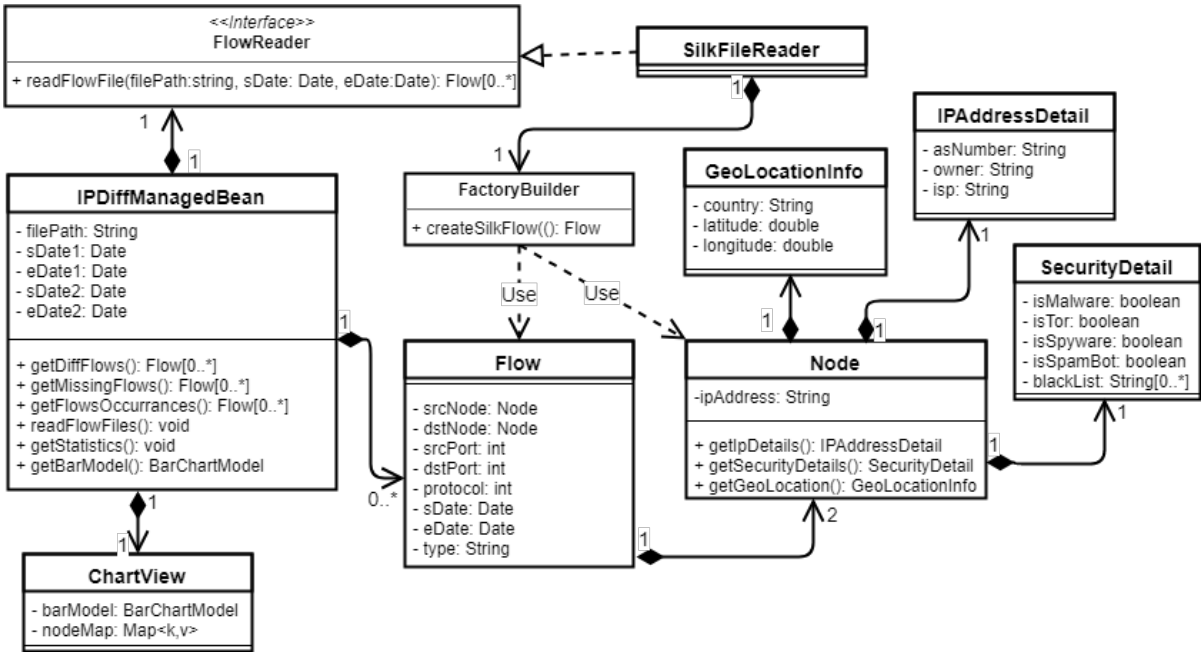


Figure 3.3: IPDiff Class Diagram

IPDiffManagedBean a bean object that handles all HTTP request parameters from/to the web browser. Upon receiving the inputs the IPDiffManagedBean object will use a **FlowReader** object to read the flow file, as specified. The FlowReader in turn delegates the flow creation process to a **Factory-Builder** object, which creates first each of the two **Node** and finally the **Flow**. The flows created are then stored for further classification.

In addition to the classes described above, other classes such as, the IPAddressDetail, SecurityDetail, ChartView, IPDiffUtils and GeoLocationInfo, may be considered as auxiliary classes used by IPDiff to accomplish its goal.

3.1.5 IPDiff Implementation

After deciding which architectural model and how to decompose the application, the next question to answer was the programming language to be used for IPDiff development. The programming language should be chosen based on skills and facilities (tools and libraries) available to achieve the goal. For that the Java programming language and the JavaServer Faces (JSF) technology were chosen. JSF is the Java Enterprise Edition (Java EE) standard for building web applications [114].

The Model View Controller (MVC) architectural pattern is used for decoupling enterprise applications' components in tree groups: (1) View defines how data are displayed to the user(user interface). (2) Model is responsible for data manipulation and storage. (3) Controller handles the user interactions by retrieving the data from the model and selecting the appropriate view to be shown for that request. MVC has been widely adopted in web application development [115, 116]. JSF applications follow the MVC pattern. In JSF framework its Java servlet component acts as the controller thus handles all http request/response data. The servlet instruct JSF GUI component for building the web pages and interacts with the Java session Bean technology that encapsulate the business logic on data access [114].

There are in the development market many applications that implements the Java EE platform. An application server may come with some or all Java EE features. The GlassFish application server was chosen for IPDiff deployment because it supports most of the Java EE technologies, e.g JSF technology, the Enterprise Java Beans (EJB) and the Java Persistence API (JPA) and comes with Grizzly web server. For the simplicity of topics covered in IPDiff neither EJB nor JPA were used.

The JSF specification provides an Application Programming Interface (API) for developing new UI components. It is here where Primeface fits in. Therefore Primeface is in simple words a library for building JSF UI components. It was designed with developers productivity in mind while keeping it lightweight for the whole application. Primeface can be added to a Java web application as a simple .jar file with no dependencies needed to be configured [117].

As of today, a wide range of ready-to-use services throughout the Web are available. The two major architectures used for designing and implementing Web services are the Remote Procedure Call (RPC) based approach and the resource-oriented approach [118]. The Representational State Transfer (REST) a resource-oriented architectural style is widely used by resource aware applications [119]. The output from a REST web service represents the status of the requested object and consists of a mapping between keys and values. The Extensible Markup Language (XML) and JavaScript Object Notation (JSON) are the most used output formats when transferring REST objects.

A good example of a company offering web services is the Google Inc. Google offers several API to web developers, such as the Google Map API [120]. The Google Map API is actually a JavaScript application that provides an interface with several methods that can be called within the local application whether it is web, desktop or mobile application. The parameters needed to customize the content to display on web pages are the latitude and the longitude of the location where the object should appear on the map. Besides those parameters other features are available, for instance, the markers. An example of tool that uses the Google Map API is the SurfMap. SurfMap uses the Google Map API to create different data visualization perspectives, which makes the network monitoring task more interesting for the network administrators [121].

There are several other companies providing web services for many different purposes. For instance, Neutrino API provides general purpose services on top of RESTfull API, such as phone, imaging, e-

commerce, geolocation and network security services, in most of case paid or with daily limit usage. The API handles more than 50 million requests a day. [122].

Mind map software is used to create diagrams that shows relationship between two connected objects that may express different concepts. Two elements are used when drawing a mind map diagram, which are, the nodes that represents the objects and an edge that shows the relationship between those nodes. Several software has implemented this concept of mind map, such as, the FreeMind widely used for expressing brainstorming session in a visual diagram, and Cmap [123].

IPDiff uses The Primeface Google Map implementation to visualize network flows end points and uses the Neutrino API for gathering security information for a given IP address. Also IPDiff will use an implementation of the concept of mind map, to map flows end points.

3.2 Flow Processing

The steps used for IPDiff flow comparison and classification are described in this section. IPDiff reads flows from two types of file format and classifies flows based on time interval.

3.2.1 Flow Files

SiLK flows are saved in binary files using a SiLK proprietary format. Thus a SiLK flow file can only be understood with SiLK tools. Therefore in my first approach to the problem, I proposed to read those files using Silk commands and redirect their output to text file, that would then be processed. In a second approach, the one that was selected, I propose to read the file in SiLK format and to process its output line by line to create the flows to be used by IPDiff.

3.2.2 Flow Classification

Filtering flows to be analysed is similar to the approach used in [76] and explained in Section 2.4.1. IPDiff flow filtering will be based on time intervals. Due to the memory constraints of the deployment machine, IPDiff implementation was made to accept a second time interval. The first interval is used just to limit the amount of flows to be considered as flows already analysed or old flows, but not for comparison, while flows that have been active during the second interval are the flows under study, which are classified as new flows. The SiLK suite *rwfilter* command when combined with the *--stime* and *--etime* filters its inputs by strict match criteria, which is not exactly what is done by IPDiff. Thus rwcut command was the chosen for reading the flows, whereas the filtering is done by IPDiff in a way that any flow being active within the time interval is considered to belong to that interval. In Figure 3.4, if $start$ and end represent the analysis interval (the second interval) and considering sf_n as start of $Flow_n$ and ef_n as end of $Flow_n$ thus $Flow_1$ and $Flow_6$ are considered not belonging to that interval while the others flows belong, so they are considered fully or partially active during the analysis interval.

Algorithm 1 is used by IPDiff to decide whether a flow belongs or not to the time interval specified by start and end.

3.2.3 Flow Information Details

The final IPDiff activity is to evaluate each flow classified as diff flow or as missing flow. Additional details about each flow are gathered from the internet. IPDiff rely on Neutrino API, to gather security information related to an IP address [122]. Neutrino provides a RESTful webservice API that accept HTTP GET or POST requests with specific parameters depending on the purpose. A HTTP POST request can be sent

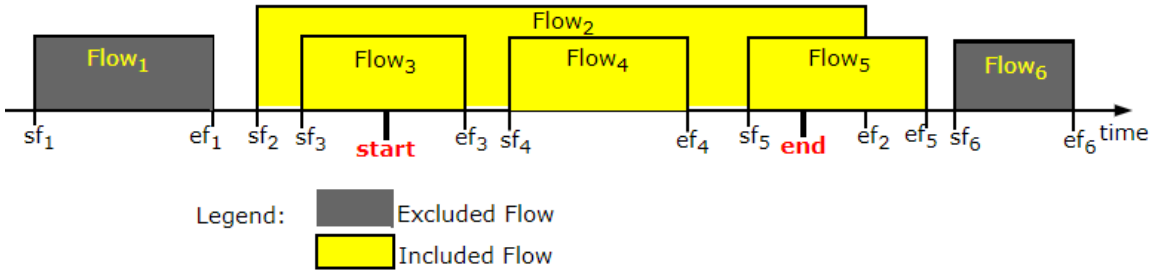


Figure 3.4: Flow Processing Time Interval

Algorithm 1: Flow Processing Based on flow start and end time

Data: start, end, flows
Result: Flows Included
foreach $flow$ *in* $flows$ **do**
  if $flowEnd$ *less than* $start$ *OR* $flowStart$ *greater than* $start$ **then**
    discard($flow$);
  if $flowStart$ *NOT greater than* $start$ *AND* $flowEnd$ *NOT greater than* end **then**
    include($flow$);
  if $flowStart$ *NOT greater than* $start$ *AND* $flowEnd$ *greater than* end **then**
    include($flow$);
  if $flowStart$ *NOT less than* $start$ *AND* $flowEnd$ *NOT greater than* end **then**
    include($flow$);
  if $flowStart$ *NOT less than* $start$ *AND* $flowEnd$ *greater than* end **then**
    include($flow$);
end

using either JSON or XML format. The same formats are available for the HTTP response. Neutrino provides many API but they have daily usage limits and all services require a registered user account for issuing the API key used for authentication purpose. For the purpose of this work, the IP Blocklist service is the most interesting because the API can classify an IP address as: Malware or Spyware, a Tor node, a Spider, a Bot or Botnets, a Spammer, an Exploit scanner [124]. The detailed output of IP BlockList service from Neutrino API can be found in Appendice A.

Similarly to the security details, geolocation information for each IP address is gathered through a web service. In this case the *ip-api.com* Geo IP API is used [125]. The API receives only HTTP GET requests with the response format and the IP address as the parameters. For response format JSON was chosen. Therefore on supplying an IP address as HTTP GET parameter and specifying JSON as the response format, the API replies with JSON object with information about IP address owner country, latitude, longitude, ISP, AS number, as shown in Figure 3.5, in case of successful query [126]. The information received from these two API are then shown in Primeface dataTable and gMap UIs.

```
response: {
  "status": "success",
  "country": "COUNTRY", "countryCode": "COUNTRY CODE", "region": "REGION CODE",
  "regionName": "REGION NAME", "city": "CITY", "zip": "ZIP CODE",
  "lat": LATITUDE, "lon": LONGITUDE, "timezone": "TIME ZONE",
  "isp": "ISP NAME", "org": "ORGANIZATION NAME", "as": "AS NUMBER / NAME",
  "query": "IP ADDRESS USED FOR QUERY"
}
```

Figure 3.5: IP-API JSON output parameters

3.3 IPDiff Deployment

This section describes how IPDiff deployment has been made, starting from the tools installation process up to the programming environment setup.

3.3.1 Deployment Environment

As already said in Section 2.5.3, The SiLK suite is only available under Linux operating systems. The Ubuntu Linux was the chosen option for its simplified mode of installation and widely tested under VirtualBox Virtual Machine (VM). These settings were chosen because they match to my personal computer hardware availability, so I could work any time and any where without even needing internet access. A 2GB memory space was allocated for the VM and fits perfectly the minimum requirements of Ubuntu 14.04 Long Term Release(LTS), as shown in [127].

3.3.2 Silk Installation

The SiLK suite is composed by many tools and daemons so for their installation a step-by-step and in order approach is recommended Depending on tools to include the installation procedure may differ. For the purpose of IPDiff the tools were installed, under Ubuntu operating system, in the following order:

1. fixbuf – Responsible for SiLK IPFIX flow processing.
2. Yaf – The SiLK flow collector.
3. SiLK – SiLK core files
4. libschematools – For SiLK flow formatting
5. Analysis Pipeline – SiLK flow processing and analysis
6. NetSA-python – python library for SiLK.

More installation details can be found in SiLK documentation [128, 129]. I also recommend to check NetSA install tools play list videos on youtube [130].

Upon following the steps suggested above, the SiLK suite tools were installed successfully on Ubuntu 14.04 LTS, running under the VirtualBox VM. The SiLK suite installed tools, directory and files naming convention can be seen in Figure 3.6.

```
ditutala@ditutala-VirtualBox: ~/tmp/SiLK-LBNL-05/in/2005/01/06
ditutala@ditutala-VirtualBox:~/tmp$ cd ~/tmp/
ditutala@ditutala-VirtualBox:~/tmp$ ls -la
total 76812
drwxrwxr-x  8 ditutala ditutala     4096 Jun 22 01:54 .
drwxr-xr-x 40 ditutala ditutala     4096 Jun 22 01:52 ..
-rw-rw-r--  1 ditutala ditutala 20722337 Mai 18 14:45 flowFile23_02.txt
drwxr-xr-x  7 ditutala ditutala     4096 Mar 18  2015 libfixbuf-1.2.0
-rw-rw-r--  1 ditutala ditutala   641195 Set 14  2012 libfixbuf-1.2.0.tar.gz
drwxrwxr-x  2 ditutala ditutala     4096 Mai  9 16:33 redTeam
drwxr-xr-x  8 ditutala ditutala     4096 Mar 18  2015 silk-2.5.0
-rw-rw-r--  1 ditutala ditutala  4467239 Jun 28  2012 silk-2.5.0.tar.gz
drwxr-xr-x  6 ditutala ditutala     4096 Fev 20 20:13 SiLK-LBNL-05
-rw-rw-r--  1 ditutala ditutala 24897458 Mar 18  2015 SiLK-LBNL-05-nonscan.tar.gz
-rw-rw-r--  1 ditutala ditutala 26500381 Mar 18  2015 SiLK-LBNL-05-scanners.tar.gz
drwxrwxr-x  2 ditutala ditutala     4096 Mai 22 12:27 textFlow
drwxr-xr-x 11 ditutala ditutala     4096 Mar 18  2015 yaf-2.3.2
-rw-rw-r--  1 ditutala ditutala  1381889 Set 14  2012 yaf-2.3.2.tar.gz
ditutala@ditutala-VirtualBox:~/tmp$ cd SiLK-LBNL-05/in/2005/01/06/
ditutala@ditutala-VirtualBox:~/tmp/SiLK-LBNL-05/in/2005/01/06$ ls
in-S0_20050106.19  in-S0_20050106.22  in-S1_20050106.20  in-S1_20050106.23
in-S0_20050106.20  in-S0_20050106.23  in-S1_20050106.21
in-S0_20050106.21  in-S1_20050106.19  in-S1_20050106.22
ditutala@ditutala-VirtualBox:~/tmp/SiLK-LBNL-05/in/2005/01/06$
```

Figure 3.6: SiLK Directory and File Structure and Naming

The Figure 3.7 demonstrate the output produced when the *rwcut* silk command is executed with the option *--num-rec=4* to display only the content of the first 4 records *--fields=1-9,21* of *in-S0-20050106.21* file.

```
ditutala@ditutala-VirtualBox:~/tmp/SiLK-LBNL-05/in/2005/01/06$ rwcut --fields=1-9,21 in-S0_20050106.21 --num-rec=4
            sIP|            dIP|sPort|dPort|pro|   packets|     bytes|   flags|                 sTime|   type|
 208.122.81.187|131.243.104.182|  110| 3640|  6|        11|       703|FS PA   |2005/01/06T21:00:40.565|     in|
192.180.211.149| 131.243.104.57|  389| 3512|  6|         7|       659|FS PA   |2005/01/06T21:01:26.779|     in|
192.180.211.149| 131.243.104.57|  389| 3513|  6|         6|       274|FS PA   |2005/01/06T21:01:27.071|     in|
192.180.211.149| 131.243.104.57|  389| 3514|  6|         6|       289|FS PA   |2005/01/06T21:01:27.247|     in|
```

Figure 3.7: SiLK rwcut Command Output

3.3.3 Programming Environment

NetBeans is one of the Java most used Integrated Development Environment (IDE). Another famous Java IDE is the Eclipse IDE. Eclipse is plugin based IDE while NetBeans comes with many tools incorporated on its installation package. For Java web development NetBeans supports much more features compared to Eclipse. Eclipse startup time is better compared to NetBeans due to the many tools that NetBeans comes with. Given that load time is not important for IPDiff, for IPDiff development NetBeans IDE 8.2 was chosen because of its built in tools [131]. The GlassFish 4.1.1 is part of those tools.

NetBeans 8.2 bundle installation process is quite easy. It has a wizard that guides the process, with step-by-step instructions up to its end. For this reason no illustration of the installation process is shown but Figure 3.8 shows NetBeans IDE 8.2 already installed and being used. The Primeface version being used is the Primeface v5.0.

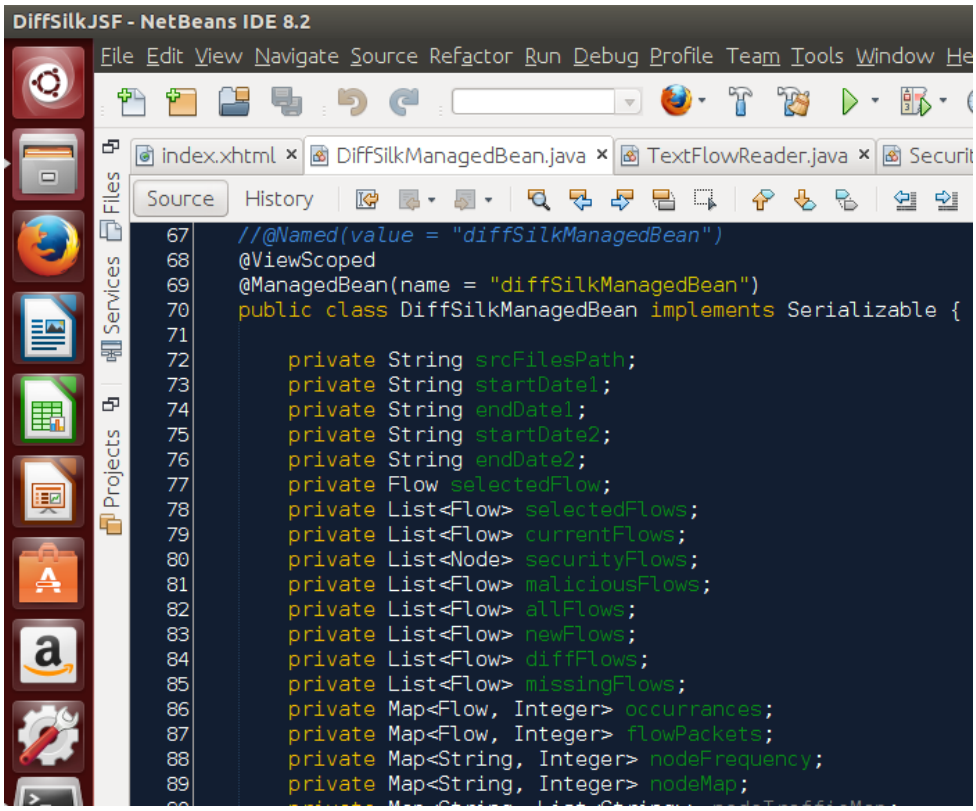


Figure 3.8: NetBeans IDE under Ubuntu OS

The detailed information on how IPDiff classes were deployed under the NetBeans web project framework, can be found in Appendice B.1.

3.3.4 IPDiff Layout Interface

The first layout used for the IPDiff entry page can be seen in Appendice B.3. The final layout is composed by a single web page, which is used to handle all IPDiff requirements. This entry page uses a tabbed layout enabled by the Primeface tabView feature. Each tab view may show IPDiff fulfilling one or more of its requirement. Also an IPDiffManagedBean object plays a role of facade which is one of design pattern widely used in the object oriented software development [132–134].

3.4 IPDiff Demonstration

In this section, IPDiff output images are shown upon receiving a click event on its entry form run button. The dates must be supplied in dateTime format: 'YY/mm/dd hh:mm:ss', as shown in Figure 3.9. The parameters are described as follow:

- File path, a string identifying the flows file name or directory.
- Old flows start date, a string representing a date.
- Old flows end date, a string representing a date.
- New flows start date, a string representing a date.
- New flows end date, a string representing a date.

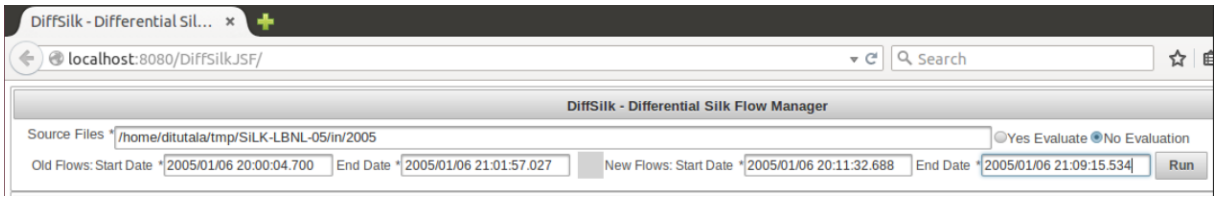
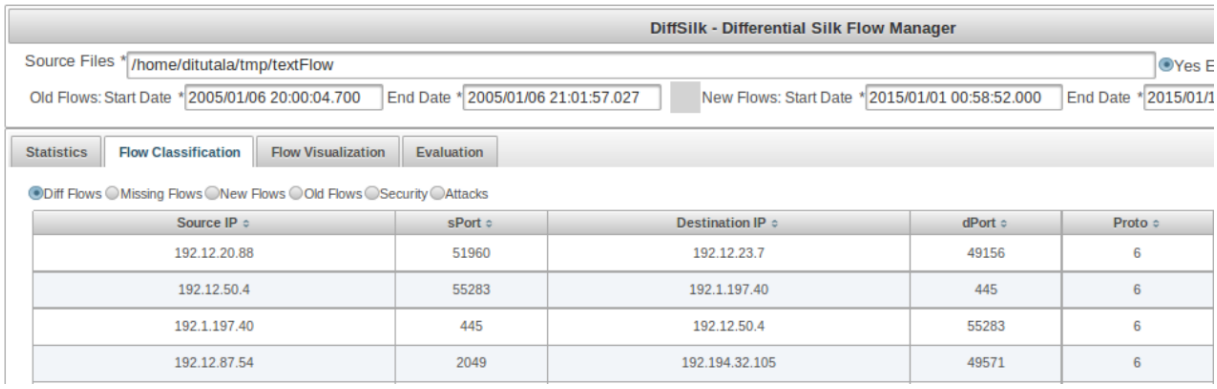


Figure 3.9: IPDiff Start Page

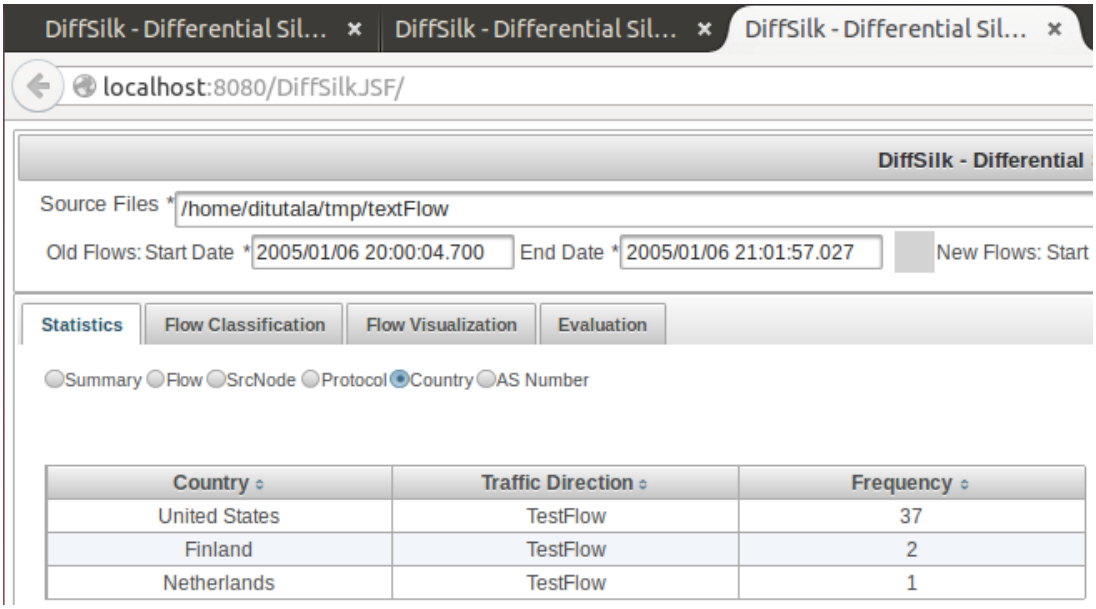
After reading flows from supplied directory the IPDiff produces flow classification which can be seen under the Flow Classification tab. A tabular output listing each flow contained on each class of flows, see Figure 3.10. To see more detail on specific flow, select it by clicking on radio button then press "More Detail". Detailed information is then shown in a Primeface growl component, which displays messages in an overlay. Statistics are provided based on flow, node, protocol, country and AS number.



Source IP	sPort	Destination IP	dPort	Proto
192.12.20.88	51960	192.12.23.7	49156	6
192.12.50.4	55283	192.1.197.40	445	6
192.1.197.40	445	192.12.50.4	55283	6
192.12.87.54	2049	192.194.32.105	49571	6

Figure 3.10: IPDiff Flow Classification

Figure 3.11 shows that processed flows are from United States of America, Finland and Netherlands. This information is gathered from the *ip-api* webservice as mentioned in Section 3.2.3.



Country	Traffic Direction	Frequency
United States	TestFlow	37
Finland	TestFlow	2
Netherlands	TestFlow	1

Figure 3.11: IPDiff - Per Country Statistics

Another output produced by IPDiff is the visualization of traffic between nodes. The MindMap Primeface UI component was the choice. The Figure 3.12 shows network flow end points, that is, flows from

the source node (blue in the midle) to several destination node (in green). Each node is identified by its IP address.

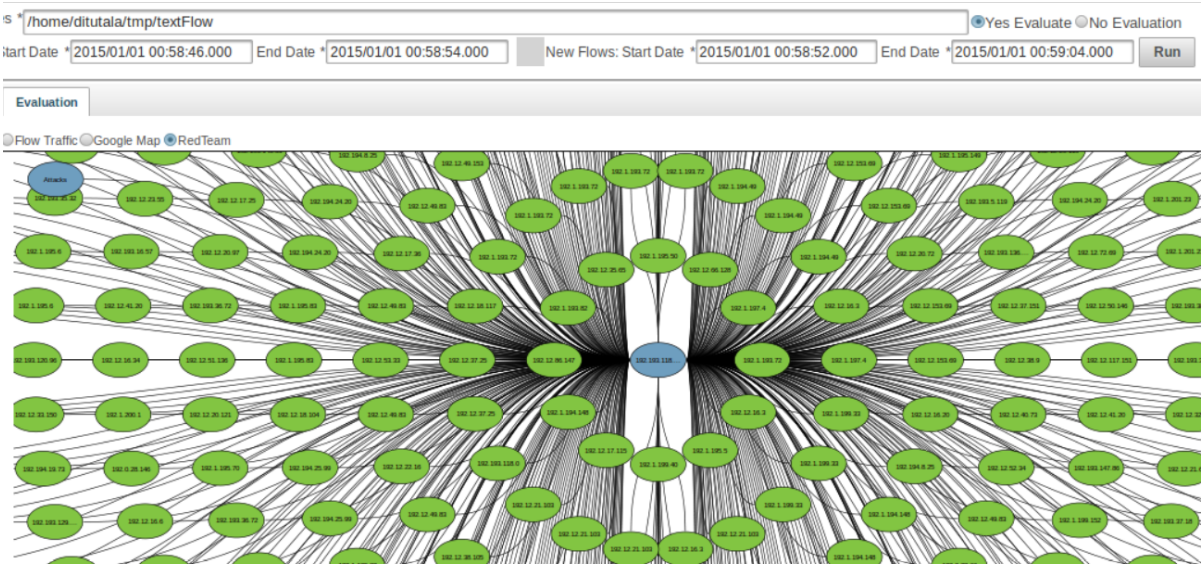


Figure 3.12: IPDiff Visualization of Node Traffic

The last image, in Figure 3.13 shows the network flow traffic on gMap Primeface utility, which is the Primeface implementation of the Google Map API. IPDiff collects geolocation information for each node IP address to build this view. In the image it can be seen that there are network traffic from/to US and traffic from US to Finland and Netherland, which confirms what was shown in the country statistics output, in Figure 3.11.

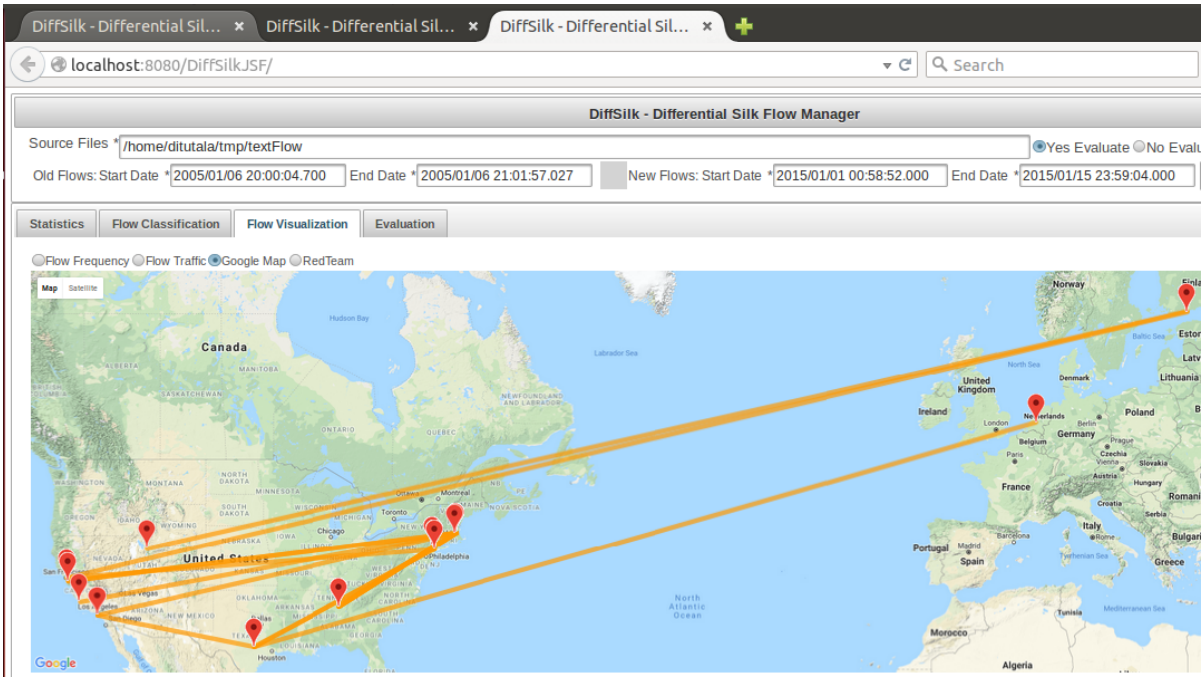


Figure 3.13: IPDiff Traffic Visualization on Google Map

3.5 Summary

In this chapter, a new tool for network change detection has been presented, which compares flows based on time intervals. The comparison results on flow classification as: (1) old flows, (2) new flows, (3) missing flows and finally (4) diff flows.

Three flow visualization models have been used and shown in this chapter. Each of them presents IPDiff results in a different perspective. To visualize summary IPDiff uses dataTable Primeface component. To see flows geolocation information a Primeface gMap component, which is a GoogleMap implementation, was used and finally to show interactions between nodes, a mindMap component was used.

Chapter 4

Evaluation

This chapter describes the evaluation process of IPDiff and discusses its main findings. To evaluate IPDiff a dataset from LANL was used. Therefore the chapter starts by showing how the dataset was processed and closes discussing the result produced by the tool.

4.1 Evaluation Dataset

To evaluate the ability of IPDiff to process flows in different formats, two datasets were used. In the first evaluation phase, the dataset used has been made available by the NetSA SiLK suite website and was produced by the Lawrence Berkeley National Laboratory (LBNL) along with the International Computer Science Institute (ICSI). The collected data represents flows occurred between 2004-10-04 and 2005-01-05 [135]. This dataset was used mostly to validate the operation of the tool, so no results about it are provided in this chapter.

In the second evaluation phase, the dataset used is publicly available at the Los Alamos National Laboratory (LANL) [136] website. This dataset is a result of collected data and made available for cyber security research. It comprises 58 days of network security events collected from five sources within their local network in 2015. The data events collected include users authentication, DNS lookups, network flows and exploits of network security threats. The dataset is in total around 12GB compressed file but can also be downloaded separately. The individual events included in the dataset are: (1) *auth.txt.gz* which contains authentication events collected from Windows-based desktop computers. (2) *proc.txt.gz* represents start stop process events also collected from Windows-based desktop computers. (3) *flows.txt.gz* contains network flows collected from central routers. (4) *dns.txt.gz* contains DNS lookups events collected on DNS server. (5) *redteam.txt.gz* which presents authentication events collected with known redteam compromised events [137].

4.1.1 File Format

For the IPDiff evaluation purpose, only the flow and redteam files were downloaded. The flow file contains 5GB of flows in text format. Each line entry represents a flow in form of (event time, duration, source computer, source port, destination computer, destination port, protocol, packet count, byte count). The redteam file, also a text file, contains flows with source IP address associated to the RedTeam users' computers. Each line represents one event in form of (event time, user@domain, source computer, destination computer).

For privacy reasons, some of the data which could be easily associated with LANL users or computers was not included and other items were de-identified, but network flows with well-known ports were

not de-identified. To keep track of users and computers, anonimized identifiers codes were introduced, e.g., C1 in all files represents the same computer and U1 represents the same user in all events in which it appears [137], as can be seen in Figure 4.1.

```
46,0,C719,N11016,C612,389,17,1,191
46,0,C719,N1733,C612,88,6,6,1765
46,0,C719,N7705,C612,88,6,5,543
46,0,C719,N7708,C612,88,6,4,417
46,10,C1905,N3336,C1685,N1,6,6,1219
46,2,C4174,137,C453,137,17,2,180
46,2,C453,137,C4174,137,17,2,156
46,38,C1015,N33,C486,N452,6,2,92
46,6,C1685,N1,C1905,N3336,6,4,2748
46,60,C1156,N892,C2588,N20,6,3,138
46,60,C1581,1433,C368,N11015,6,9,1015
46,60,C2588,N20,C1156,N892,6,3,138
46,60,C368,N11015,C1581,1433,6,11,1112
47,0,C1173,445,C1220,N576,6,1,46
47,0,C1220,N576,C1173,445,6,1,52
47,0,C1707,N1,C4135,N10712,6,5,383
```

(a) LANL Flow File Content

```
150885,U620@DOM1,C17693,C1003
151036,U748@DOM1,C17693,C305
151648,U748@DOM1,C17693,C728
151993,U6115@DOM1,C17693,C1173
153792,U636@DOM1,C17693,C294
155219,U748@DOM1,C17693,C5693
155399,U748@DOM1,C17693,C152
155460,U748@DOM1,C17693,C2341
155591,U748@DOM1,C17693,C332
156658,U748@DOM1,C17693,C4280
210086,U748@DOM1,C18025,C1493
210294,U748@DOM1,C18025,C1493
210312,U748@DOM1,C18025,C1493
218418,U748@DOM1,C17693,C504
227052,U748@DOM1,C17693,C148
227408,U748@DOM1,C17693,C148
```

(b) LANL RedTeam File Content

Figure 4.1: LANL Dataset Files Used

4.1.2 Flow Processing

The flows contained in *flows.txt.gz* differ from network flows so the file must be processed line by line to retrieve the field values required to compose a network flow. Another and very important issue is the flow source and destination IP addresses and ports which were coded. To solve this problem, padding was used where necessary when converting the information read from each line. In flow file, the computers identifiers codes used have variable length. For instance, there are cases of codes with only 2 characters and others with 6 characters. On the other hand, the Java InetAddress method used to convert a string IP address to its byte representation fails if the string passed through its arguments does not satisfy its requirements, that is, can not be convert to a byte. To overcome this problem, a variable length hexadecimal string was used for padding to the input according to its length, as shown in Listing 4.1.

```
public static String getHex2IPAddress(String str) throws UnknownHostException {
  String[] padds = {"", "C", "C0", "C00", "C001", "C0001", "C00001", "C000001"};
  InetAddress ip;
  int len = str.length();
  if (len > 0 && len <= 8) {
    String ipStr = padds[8 - len] + str.trim();

    ip = InetAddress.getByAddress(DatatypeConverter.parseHexBinary(ipStr));
    return ip.getHostAddress();
  }
  return "127.0.0.1";
}
```

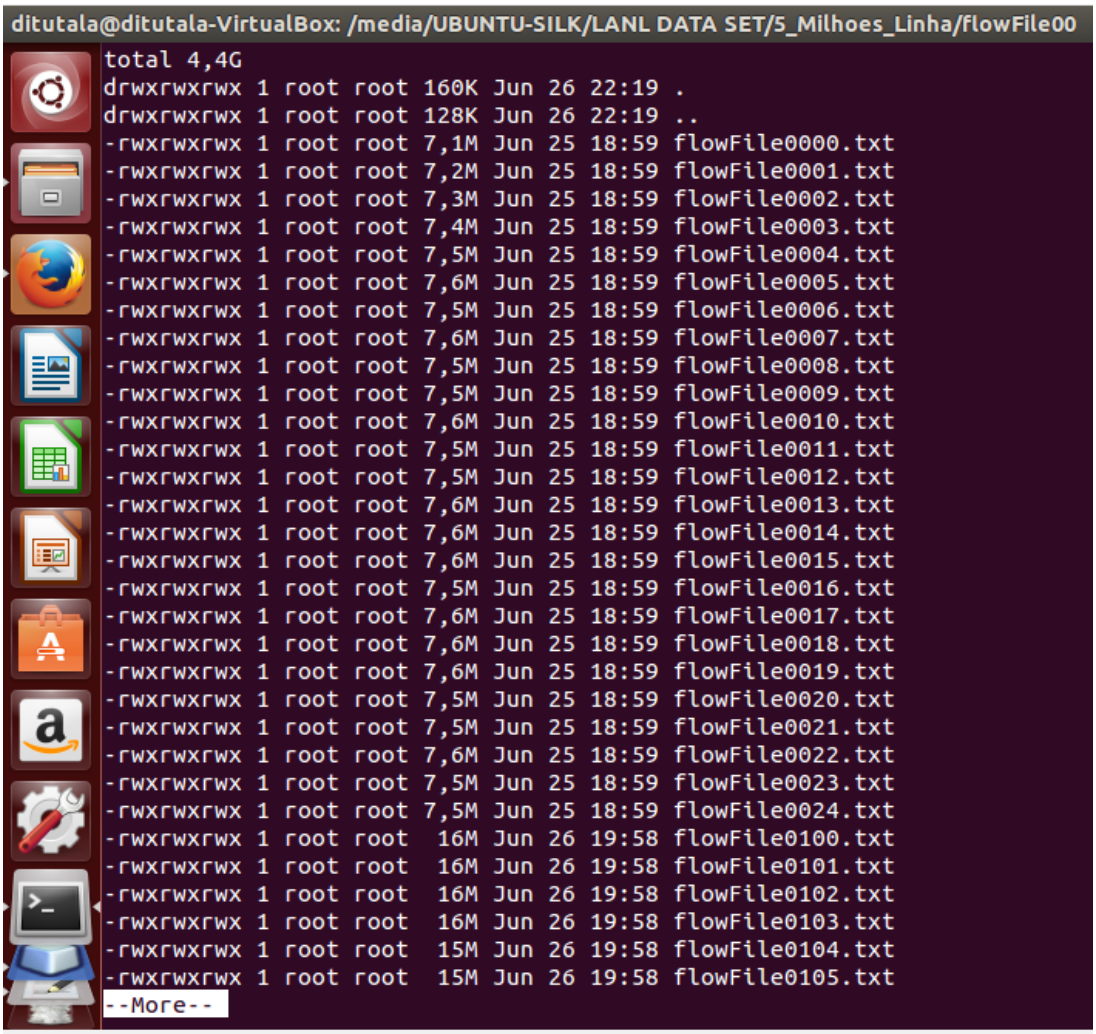
Listing 4.1: IP Address Padding and Conversion

A similar procedure was used to determine a flow start and end time, see Listing D.1 for more details. In the files, events date are represented in seconds. At this point to get the value for flow end time, given

its start time a 1 second was added to start time, for events with 0s duration. As for a time reference, the 2015 January the first was chosen, for any particular reason.

4.2 Experimental Results and Discussion

To conduct the experiments the flow file was split into smaller files due to memory limitation of the virtual machine used to drive the tests, as can be seen in Figure 4.2. This results on files with different sizes which were used.



```
ditutala@ditutala-VirtualBox: /media/UBUNTU-SILK/LANL DATA SET/5_Milhoes_Linha/flowFile00
total 4,4G
drwxrwxrwx 1 root root 160K Jun 26 22:19 .
drwxrwxrwx 1 root root 128K Jun 26 22:19 ..
-rwxrwxrwx 1 root root 7,1M Jun 25 18:59 flowFile0000.txt
-rwxrwxrwx 1 root root 7,2M Jun 25 18:59 flowFile0001.txt
-rwxrwxrwx 1 root root 7,3M Jun 25 18:59 flowFile0002.txt
-rwxrwxrwx 1 root root 7,4M Jun 25 18:59 flowFile0003.txt
-rwxrwxrwx 1 root root 7,5M Jun 25 18:59 flowFile0004.txt
-rwxrwxrwx 1 root root 7,6M Jun 25 18:59 flowFile0005.txt
-rwxrwxrwx 1 root root 7,5M Jun 25 18:59 flowFile0006.txt
-rwxrwxrwx 1 root root 7,6M Jun 25 18:59 flowFile0007.txt
-rwxrwxrwx 1 root root 7,5M Jun 25 18:59 flowFile0008.txt
-rwxrwxrwx 1 root root 7,5M Jun 25 18:59 flowFile0009.txt
-rwxrwxrwx 1 root root 7,6M Jun 25 18:59 flowFile0010.txt
-rwxrwxrwx 1 root root 7,5M Jun 25 18:59 flowFile0011.txt
-rwxrwxrwx 1 root root 7,5M Jun 25 18:59 flowFile0012.txt
-rwxrwxrwx 1 root root 7,6M Jun 25 18:59 flowFile0013.txt
-rwxrwxrwx 1 root root 7,6M Jun 25 18:59 flowFile0014.txt
-rwxrwxrwx 1 root root 7,6M Jun 25 18:59 flowFile0015.txt
-rwxrwxrwx 1 root root 7,5M Jun 25 18:59 flowFile0016.txt
-rwxrwxrwx 1 root root 7,6M Jun 25 18:59 flowFile0017.txt
-rwxrwxrwx 1 root root 7,6M Jun 25 18:59 flowFile0018.txt
-rwxrwxrwx 1 root root 7,6M Jun 25 18:59 flowFile0019.txt
-rwxrwxrwx 1 root root 7,5M Jun 25 18:59 flowFile0020.txt
-rwxrwxrwx 1 root root 7,5M Jun 25 18:59 flowFile0021.txt
-rwxrwxrwx 1 root root 7,6M Jun 25 18:59 flowFile0022.txt
-rwxrwxrwx 1 root root 7,5M Jun 25 18:59 flowFile0023.txt
-rwxrwxrwx 1 root root 7,5M Jun 25 18:59 flowFile0024.txt
-rwxrwxrwx 1 root root  16M Jun 26 19:58 flowFile0100.txt
-rwxrwxrwx 1 root root  16M Jun 26 19:58 flowFile0101.txt
-rwxrwxrwx 1 root root  16M Jun 26 19:58 flowFile0102.txt
-rwxrwxrwx 1 root root  16M Jun 26 19:58 flowFile0103.txt
-rwxrwxrwx 1 root root  15M Jun 26 19:58 flowFile0104.txt
-rwxrwxrwx 1 root root  15M Jun 26 19:58 flowFile0105.txt
--More--
```

Figure 4.2: IPDiff Dataset Split

The data used for all experiments can be seen in the Figure 4.3.

#	File	FileSize[MB]	Start Eval. Interval	End Eval. Interval	ElapsedTime[s]	FileSize[MB]	OldFlows	NewFlows	DiffFlows	MissingFlows
1	flowFile1408.txt	16	2015/01/09 09:20:00.000	2015/01/09 09:30:00.000	55,488	16	78888	13012	0	0
2	flowFile1408.txt	16	2015/01/09 09:20:00.000	2015/01/09 09:30:00.000	40,761	16	78888	13012	0	0
3	flowFile1408.txt	16	2015/01/09 09:20:00.000	2015/01/09 09:35:00.000	99,121	16	157776	26024	0	0
4	flowFile1409.txt	16	2015/01/09 09:55:00.000	2015/01/09 10:30:00.000	17,1475	16	207172	181660	4003	19252
5	flowFile1409.txt	16	2015/01/09 09:55:00.000	2015/01/09 10:50:00.000	11,5935	16	424934	399422	4003	19252
6	flowFile150.txt	47	2015/01/09 13:50:00.000	2015/01/09 14:00:00.000	195,58	47	87774	70800	0	0
7	flowFile150.txt	47	2015/01/09 13:50:00.000	2015/01/09 14:30:00.000	151,853	47	228142	280830	34929	8283
8	flowFile2112.txt	7,9	2015/01/14 09:50:00.000	2015/01/14 10:11:45.000	13,28	7,9	107453	154987	50312	23562
9	flowFile2100.txt	16	2015/01/13 18:20:00.000	2015/01/13 18:46:06.000	19,555	16	320493	357277	39359	19969
10	flowFile2200.txt	16	2015/01/14 10:20:00.000	2015/01/14 11:10:00.000	18,882	16	132058	342271	157786	37051
11	flowFile2012.txt	16	2015/01/13 15:45:00.000	2015/01/13 16:44:30.000	33,618	16	0	93	93	0
12	flowFile2012.txt	16	2015/01/13 16:50:00.000	2015/01/13 17:21:59.000	21,487	16	87054	167788	59492	18423
13	flowFile20.txt	32	2015/01/13 09:35:00.000	2015/01/13 09:50:30.000	102,643	32	99127	105933	38373	32756
14	flowFile20.txt	32	2015/01/13 09:30:00.000	2015/01/13 10:00:30.000	78,448	32	99127	208822	78932	14583
15	flowFile21.txt	48	2015/01/13 11:50:00.000	2015/01/13 12:30:00.000	110,236	48	30063	243426	183033	0
16	flowFile2207.txt	16	2015/01/14 17:00:00.000	2015/01/14 17:48:45.000	17,464	16	158996	208523	205097	156944
17	flowFile2207.txt	16	2015/01/14 17:30:00.000	2015/01/14 17:51:00.000	169,02	16	108061	81188	79730	106708

Figure 4.3: Experimental data

The first experiments made, no flows were detected belonging to the first interval, so all flows has been classified as new flows. Since there were no old flows, the number of diff flows should be equal to the number of new flows, as can be seen in Figure 4.4a. In the opposite direction IPDiff will not detect diff flows if all new flows are already known, thus being part of old flows, as shown in Figure 4.4b.

Flow Processing Time	
Elapsed Time	33.618s

Flow Statistics	
CATEGORY	TOTAL
Old Flows	0.0
New Flows	93.0
Diff Flows	93.0
Missing Flows	0.0
DiffFlows Attacks	0.0
MissingFlows Attacks	0.0

(a) No Old Flows Found

Flow Processing Time	
Elapsed Time	99.121s

Flow Statistics	
CATEGORY	TOTAL
Old Flows	157776.0
New Flows	26024.0
Diff Flows	0.0
Missing Flows	0.0
Attacks	0.0

(b) No diff flows Found

Figure 4.4: Experimental Results from two runs with two files

IPDiff was developed with the aim of detecting any new and unknown network flow based on time interval. In total, 17 experiments were made and the results suggest that IPDiff could detect diff flows and missing flows depending on data interval and file chosen in almost all the experiments, as can be seen in Figure 4.5. The output in Figure 4.5a shows that no diff flows were found even though, the run took about 99s and produced about 157776 flows as old flows, while over 26024 were classified as new flows. In the next Figure 4.5b about 4003 flows were classified as diff flows. Therefore the first interesting outcome from this experiments, spot that the IPDiff flow processing time depends on the number of flows found in a given file and the time interval chosen, even when processing files with the same size.

Flow Processing Time	
Elapsed Time	99.121s
Flow Statistics	
CATEGORY	TOTAL
Old Flows	157776.0
New Flows	26024.0
Diff Flows	0.0
Missing Flows	0.0
Attacks	0.0

(a) No Diff flows and Missing flows found

Flow Processing Time	
Elapsed Time	171.475s
Flow Statistics	
CATEGORY	TOTAL
Old Flows	207172.0
New Flows	181660.0
Diff Flows	4003.0
Missing Flows	19252.0
DiffFlows Attacks	0.0
MissingFlows Attacks	0.0

(b) Diff flows and Missing flows found

Figure 4.5: Experimental Results from two runs with two files

Another aim of IPDiff was to find out detailed security information about a particular flow that belongs to the diff flows. The Figure 4.6 shows that in addition to detecting new flows and classifying them as diff flows or missing flows, IPDiff could also detect, among the diff flows, 10 flows from the redteam users, which in case suggest a threat. If real data were used this would represent an important information to network management security awareness.

Flow Processing Time	
Elapsed Time	115.935s
Flow Statistics	
CATEGORY	TOTAL
Old Flows	424934.0
New Flows	399422.0
Diff Flows	4003.0
Missing Flows	19252.0
DiffFlows Attacks	10.0
MissingFlows Attacks	0.0

Figure 4.6: Diff flows and Missing Flows Detected

The IPDiff performance, in a different perspective, can be seen in Figure 4.7. As already said the file split process has produced several small files. In the same image it is possible to see 10 run with 16MB files has taken 10 different computation time. The maximum elapsed time occurred while using a file with 47MB of flows. This maximum elapsed time might have occurred due to the amount of new flows processed in that experiments time interval. It is also interesting to spot that there was an experiment that resulted on memory exception, which can be confirmed in the same image, with elapsed time of 0.

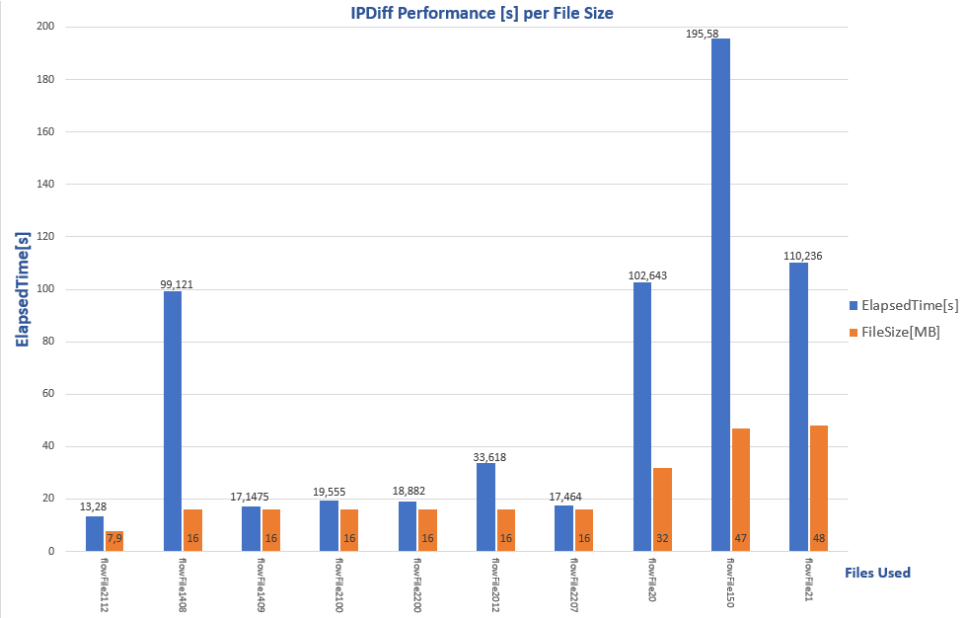
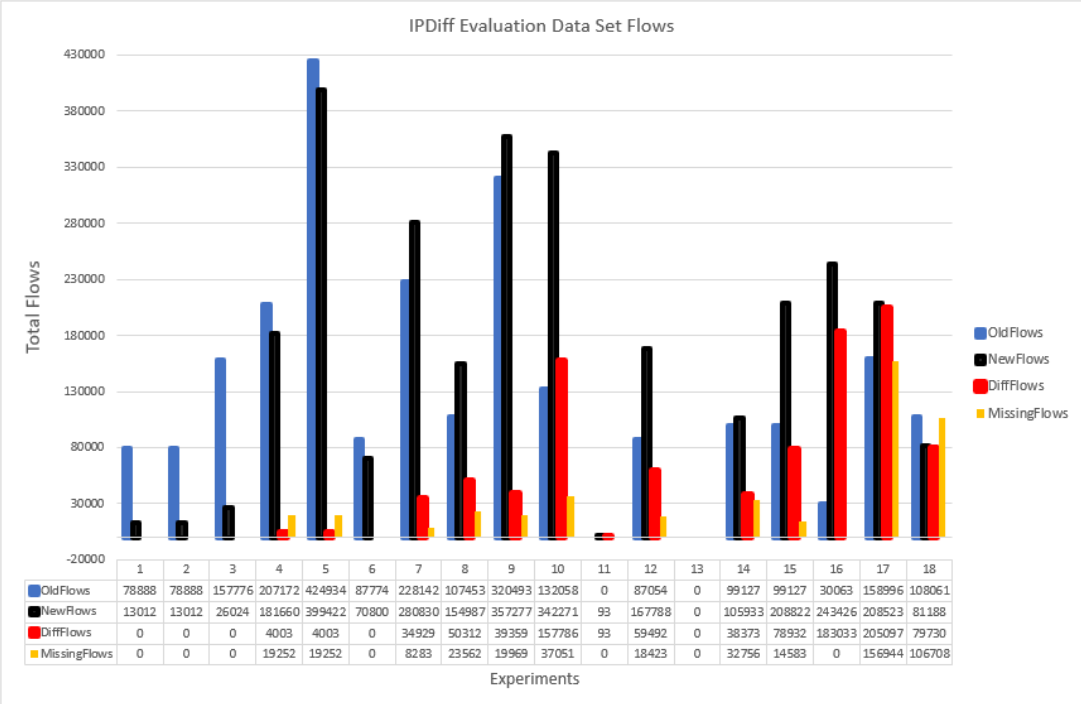


Figure 4.7: IPDiff Performance Evaluation

The Figure 4.8 provides a summary of the experimental results obtained during the IPDiff evaluation process. The size of the evaluation time intervals were selected randomly. The number of new flows depends on whether or not a chosen file contains flows on the selected time interval. The same apply to old flows. As for the diff flows and missing flows they are obtained as already explained in previous sections.

From the same chart, it is possible to conclude that the number of diff flows has non relation to the number of old flows or new flows found in that interval. But a closer look to the chart shows that the number of diff flow found is greater than the number of missing flows in all cases where the number of new flows is greater than the number of old flows, but this does not hold in the opposite direction. Therefore, the results suggest non dependency of flows to the time intervals but with both time interval and the file. To sum up, the maximum number of diff flows found was about 205097 in a run that took about 17s, while processing a flow file of 16MB. Using the same file (see Figure 4.4), it can be seen that a small change in the time interval has led to a significant decrease in the number of diff flows detected. Therefore, when using LANL dataset it was clear that the main challenge of splitting the flow file was the match between the resulting smaller files and the evaluation interval. Even though, the overall IPDiff goal was fulfilled.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
OldFlows	78888	78888	157776	207172	424934	87774	228142	107453	320493	132058	0	87054	0	99127	99127	30063	158996	108061
NewFlows	13012	13012	26024	181660	399422	70800	280830	154987	357277	342271	93	167788	0	105933	208822	243426	208523	81188
DiffFlows	0	0	0	4003	4003	0	34929	50312	39359	157786	93	59492	0	38373	78932	183033	205097	79730
MissingFlows	0	0	0	19252	19252	0	8283	23562	19969	37051	0	18423	0	32756	14583	0	156944	106708

Figure 4.8: Experimental Results Summary

An attempt to show the maximum diff flows, by using different flow files with different time interval, has shown that all were considered as missing flows and among the diff flows 35 of them were also found within redteam flows, see Figure 4.9.

Flow Processing Time	
Elapsed Time	23.65s
Flow Statistics	
CATEGORY	TOTAL
Old Flows	17.0
New Flows	500000.0
Diff Flows	500000.0
Missing Flows	17.0
Attacks	35.0

Figure 4.9: Maximum diff flows processed

4.3 Summary

As mentioned in the literature review, using differences to detect new network traffic seems to be a good approach. In IPDiff this was clear. When computing changes only over diff flows IPDiff response time was faster on fewer number of flows and slower in presence of huge number of flows despite of the size of the source flow file used, as can be seen in Figure 4.4.

Although machine learning supervised or unsupervised techniques were not used in IPDiff as in [90], Figure 4.8 shows that IPDiff was able to detect and classify different flows when processing the LANL flows dataset.

Overall, the evaluation results suggest that IPDiff has fulfilled all of its design requirements since that IPDiff, has shown, could read flows in the SiLK format and also raw text file format. Meanwhile IPDiff could compare and detected differences based on time intervals. Finally IPDiff has visualized those changes in three different formats.

Chapter 5

Conclusion

This thesis presents a tool capable of using network flow information to detect changes in the network and identify their sources with focus on answering questions such as what happened on network? Which devices were involved? Are we under attack?. This kind of questions rises when problems occur. IPDiff experimental results suggest that by using network flows network management tools are capable of answering those questions.

From the literature review conducted and the experimental results obtained I conclude that:

- Several application are using the SiLK suite for network flow processing
- When processing network flows, the computation time depends on amount of flows processed
- With network flows it is easy to produce summaries of network traffic
- The experimental results have shown that IPDiff was able to read network flows from the SiLK suite, compare them based on time interval and visualized the differences

Machine learning supervised and unsupervised algorithms are among top researches when developing a network flow detection and classification application. IPDiff does not include machine learning capabilities. Therefore, an approach towards making IPDiff a more useful tool for network management, would be to improve it with such capabilities. Moreover a persistence mechanism for storing processed flows will add positive impact on the IPDiff performance.

Bibliography

[1] Y. Benkler, *The wealth of networks: How social production transforms markets and freedom*. Yale University Press, 2006.

[2] A. Sophia van Zyl, "The impact of social networking 2.0 on organisations," *The Electronic Library*, vol. 27, no. 6, pp. 906–918, 2009.

[3] IANA, "Service Name and Transport Protocol Port Number Registry." [Online]. Available: `http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml`. Accessed on 2015-01-08.

[4] M. Collins, *Network Security Through Data Analysis: Building Situational Awareness*. O'Reilly, 2014.

[5] Malkin, G., "Traceroute using an IP option." [Online]. Available: `https://tools.ietf.org/html/rfc1393`. Accessed on 2018-06-24.

[6] Z. Škiljan and B. Radic, "Monitoring systems: Concepts and tools," in *The Six CARNET Users Conference*, 2004.

[7] T. Oetiker and D. Rand, "Mrtg: The multi router traffic grapher.," in *LISA*, vol. 98, pp. 141–148, 1998.

[8] G. Nascimento and M. Correia, "Anomaly-based intrusion detection in software as a service," in *Dependable Systems and Networks Workshops (DSN-W), 2011 IEEE/IFIP 41st International Conference on*, pp. 19–24, IEEE, 2011.

[9] CERT NetSA, "Monitoring for Large-Scale Networks." [Online]. Available: `https://tools.netsa.cert.org/silk/`. Accessed on 2018-06-24.

[10] T. Taylor, D. Paterson, J. Glanfield, C. Gates, S. Brooks, and J. McHugh, "Flovis: Flow visualization system," in *Conference For Homeland Security, 2009. CATCH'09. Cybersecurity Applications & Technology*, pp. 186–198, IEEE, 2009.

[11] W. Yurcik, "Visualizing netflows for security at line speed: The sift tool suite.," in *LISA*, pp. 169–176, 2005.

[12] J. R. Goodall and M. Sowul, "VIAssist: Visual analytics for cyber defense," in *Technologies for Homeland Security, 2009. HST'09. IEEE Conference on*, pp. 143–150, IEEE, 2009.

[13] D. Plonka, "Flowscan: A network traffic flow reporting and visualization tool.," in *LISA*, pp. 305–317, 2000.

[14] R. Berthier, M. Cukier, M. Hiltunen, D. Kormann, G. Vesonder, and D. Sheleheda, "Nfsight: netflow-based network awareness tool," in *Proceedings of LISA'10: 24th Large Installation System Administration Conference*, p. 119, 2010.

[15] J. R. Quinlan, *C4. 5: programs for machine learning*. Elsevier, 2014.

[16] A. Sperotto, G. Schaffrath, R. Sadre, C. Morariu, A. Pras, and B. Stiller, "An overview of IP flow-based intrusion detection.," *IEEE Communications Surveys and Tutorials*, vol. 12, no. 3, pp. 343–356, 2010.

[17] A. A. Mahimkar, H. H. Song, Z. Ge, A. Shaikh, J. Wang, J. Yates, Y. Zhang, and J. Emmons, "Detecting the performance impact of upgrades in large operational networks," in *ACM SIGCOMM Computer Communication Review*, vol. 40, pp. 303–314, ACM, 2010.

[18] A. Mahimkar, Z. Ge, J. Wang, J. Yates, Y. Zhang, J. Emmons, B. Huntley, and M. Stockert, "Rapid detection of maintenance induced changes in service performance," in *Proceedings of the Seventh COnference on emerging Networking EXperiments and Technologies*, p. 13, ACM, 2011.

[19] K. Tgavalekos, J. M. Namayanja, and R. Alhassan, "Characterization of network behavior to detect changes: A cybersecurity perspective," in *Proceedings of the Workshop Program of the 19th International Conference on Distributed Computing and Networking*, Workshops ICDCN '18, (New York, NY, USA), pp. 2:1–2:6, ACM, 2018.

[20] J. W. Hunt and M. D. MacIlroy, *An algorithm for differential file comparison*. Bell Laboratories Murray Hill, 1976.

[21] T. Saydam and T. Magedanz, "From networks and network management into service and service management," *Journal of Network and Systems Management*, vol. 4, no. 4, pp. 345–348, 1996.

[22] E. Kuhn, "Google apologizes, explains Gmail outage." `http://edition.cnn.com/2009/TECH/09/01/gmail.outage/index.html`, accessed 2018-05-29.

[23] R. Miller, "Why Does Gmail Go Down." `http://www.datacenterknowledge.com/archives/2012/12/10/why-does-gmail-go-down`, accessed 2018-05-29.

[24] Lee Mathews, "Boeing Is The Latest WannaCry Ransomware Victim." [Online]. Available: `https://www.forbes.com/sites/leemathews/2018/03/30/boeing-is-the-latest-wannacry-ransomware-victim/#5caa0b7e6634`. Accessed on 2018-06-24.

[25] Lanxon, Nate and Ross, Tim, "U.K. Blames North Korea for WannaCry Attack on Health Service." [Online]. Available: `https://www.bloomberg.com/news/articles/2017-10-26/u-k-government-watchdog-slams-health-service-over-wannacry`. Accessed on 2018-06-24.

[26] J. D. Case, M. Fedor, M. L. Schoffstall, and C. Davin, "RFC 1157: Simple network management protocol (SNMP)," May 1990. See also STD0015 [**?**]. Obsoletes RFC1098 [**?**]. Status: STANDARD.

[27] M. T. Rose, "Management information base for network management of TCP/IP-based internets: MIB-II," *Management*, 1990.

[28] Rose, M., "Convention for defining traps for use with the SNMP." [Online]. Available: `https://tools.ietf.org/html/rfc1215`. Accessed on 2018-06-24.

[29] U. Blumenthal and B. Wijnen, "User-based security model (USM) for version 3 of the simple network management protocol (SNMPv3)," RFC 2274, RFC Editor, January 1998.

[30] Waldbusser, Steve, "Remote network monitoring management information base version 2." [Online]. Available: `https://tools.ietf.org/html/rfc4502`. Accessed on 2018-06-24.

[31] Gerhards, R. and GmbH, A., "The Syslog Protocol." [Online]. Available: `https://tools.ietf.org/html/rfc5424`. Accessed on 2018-06-24.

[32] R. Accorsi, "Safe-keeping digital evidence with secure logging protocols: State of the art and challenges," in *IT Security Incident Management and IT Forensics, 2009. IMF'09. Fifth International Conference on*, pp. 94–110, IEEE, 2009.

[33] T. Ylönen, "SSH: Secure login connections over the internet," in *Proceedings of the 6th Conference on USENIX Security Symposium, Focusing on Applications of Cryptography - Volume 6*, SSYM'96, (Berkeley, CA, USA), pp. 4–4, USENIX Association, 1996.

[34] Ylonen, Tatu and Lonvick, Chris, "The secure shell SSH protocol architecture." [Online]. Available: `https://www.rfc-editor.org/info/rfc4251`. Accessed on 2018-06-24.

[35] D. Schatzmann, W. Mühlbauer, T. Spyropoulos, and X. Dimitropoulos, "Digging into HTTPS: Flow-based classification of webmail traffic," in *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, IMC '10, (New York, NY, USA), pp. 322–327, ACM, 2010.

[36] J.-K. Hong, S.-S. Kwon, and J.-Y. Kim, "WebTrafMon: Web-based internet/intranet network traffic monitoring and analysis system," *Computer Communications*, vol. 22, no. 14, pp. 1333–1342, 1999.

[37] D. Wong, C. Ting, and C. Yeh, "From network management to service management-a challenge to telecom service providers," in *Innovative Computing, Information and Control, 2007. ICICIC'07. Second International Conference on*, pp. 280–280, IEEE, 2007.

[38] R. Boutaba and J. Xiao, "Network management: State of the art," in *Communication Systems*, pp. 127–145, Springer, 2002.

[39] Schoenwaelder, Juergen, "Overview of the 2002 IAB network management workshop." [Online]. Available: `https://tools.ietf.org/html/rfc3535`. Accessed on 2018-06-24.

[40] A. Bieszczad, P. K. Biswas, W. Buga, M. Malek, and H. Tan, "Management of heterogeneous networks with intelligent agents," *Bell Labs technical journal*, vol. 4, no. 4, pp. 109–135, 1999.

[41] Case, Jeffrey and McCloghrie, Keith and Rose, M and Waldbusser, Steven, "Coexistence between version 1 and version 2 of the Internet-standard Network Management Framework." [Online]. Available: `https://tools.ietf.org/html/rfc3584`. Accessed on 2018-06-24.

[42] M. Kahani and H. Beadle, "Decentralised approaches for network management," *ACM SIGCOMM Computer Communication Review*, vol. 27, no. 3, pp. 36–47, 1997.

[43] J. W.-K. Hong, J.-Y. Kong, T.-H. Yun, J.-S. Kim, J.-T. Park, and J.-W. Baek, "Web-based intranet services and network management," *IEEE Communications Magazine*, vol. 35, no. 10, pp. 100–110, 1997.

[44] J. Pavon, J. Tomas, Y. Bardout, and L.-H. Hauw, "CORBA for network and service management in the TINA framework," *IEEE Communications Magazine*, vol. 36, no. 3, pp. 72–79, 1998.

[45] R. M. Mortier, P. Barham, and R. Isaacs, “Distributed network management,” Dec. 13 2011. US Patent 8,077,718.

[46] J.-P. Martin-Flatin, S. Znaty, and J.-P. Hubaux, “A survey of distributed enterprise network and systems management paradigms,” *Journal of Network and Systems Management*, vol. 7, no. 1, pp. 9–26, 1999.

[47] J.-P. Martin-Flatin, “Push vs. pull in web-based network management,” in *Integrated Network Management, 1999. Distributed Management for the Networked Millennium. Proceedings of the Sixth IFIP/IEEE International Symposium on*, pp. 3–18, IEEE, 1999.

[48] C. J. Carcerano, J. D. Barnard, R. A. Wilson Jr, and D. P. Gibson, “Browser-based network management allowing administrators to use web browser on user’s workstation to view and update configuration of network devices,” Oct. 23 2001. US Patent 6,308,205.

[49] A. Conta, S. Deering, and M. Gupta, “Internet control message protocol (icmpv6) for the internet protocol version 6 (ipv6) specification,” RFC 4443, RFC Editor, March 2006. `http://www.rfc-editor.org/rfc/rfc4443.txt`.

[50] J. Kurose and K. Ross, *Computer Networking: A Top-Down Approach*, p. 379. England: Pearson, 6th ed., 2013.

[51] Perkins, Charles, “IP encapsulation within IP.” [Online]. Available: `https://www.rfc-editor.org/rfc/pdfrfc/rfc2003.txt.pdf`. Accessed on 2018-06-24.

[52] M.-S. Kim, H.-J. Kong, S.-C. Hong, S.-H. Chung, and J. W. Hong, “A flow-based method for abnormal network traffic detection,” in *Network operations and management symposium, 2004. NOMS 2004. IEEE/IFIP*, vol. 1, pp. 599–612, IEEE, 2004.

[53] Z. M. Mao, J. Rexford, J. Wang, and R. H. Katz, “Towards an accurate AS-level traceroute tool,” in *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pp. 365–378, ACM, 2003.

[54] T.-F. Yen, A. Oprea, K. Onarlioglu, T. Leetham, W. Robertson, A. Juels, and E. Kirda, “Beehive: Large-scale log analysis for detecting suspicious activity in enterprise networks,” in *Proceedings of the 29th Annual Computer Security Applications Conference*, pp. 199–208, ACM, 2013.

[55] D. Gonçalves, J. Bota, and M. Correia, “Big data analytics for detecting host misbehavior in large logs,” in *Trustcom/BigDataSE/ISPA, 2015 IEEE*, vol. 1, pp. 238–245, IEEE, 2015.

[56] D. R. Kerr and B. L. Bruins, “Network flow switching and flow data export,” June 5 2001. US Patent 6,243,667.

[57] Claise, Benoit, “Cisco systems netflow services export version 9.” [Online]. Available: `https://www.rfc-editor.org/info/rfc3954`. Accessed on 2018-06-24.

[58] Cisco, “Introduction to Cisco IOS NetFlow - A Technical Overview.” [Online]. Available: `https://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/ios-netflow/prod_white_paper0900aecd80406232.html?dtid=osscdc000283`. Accessed on 2018-06-19.

[59] J. Quittek, T. Zseby, B. Claise, and S. Zander, “Requirements for IP flow information export (IPFIX),” RFC 3917, RFC Editor, October 2004.

[60] A. C. Myers, "Jflow: Practical mostly-static information flow control," in *Proceedings of the 26th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pp. 228–241, ACM, 1999.

[61] Huawei Technologies Co., Ltd, "Huawei NetStream Traffic Statistics and Analysis White Paper." [Online]. Available: `http://e.huawei.com/en/material/onLineView?materialid=f2d3e2fc3299409eb397ef6c73fb3910`. Accessed on 2018-06-24.

[62] P. Phaal, S. Panchen, and N. McKee, "InMon corporation's sFlow: A method for monitoring traffic in switched and routed networks," RFC 3176, RFC Editor, September 2001. `http://www.rfc-editor.org/rfc/rfc3176.txt`.

[63] R. Hofstede, P. Čeleda, B. Trammell, I. Drago, R. Sadre, A. Sperotto, and A. Pras, "Flow monitoring explained: From packet capture to data analysis with netflow and ipfix," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 4, pp. 2037–2064, 2014.

[64] P. Velan, "Practical experience with IPFIX flow collectors," in *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*, pp. 1021–1026, IEEE, 2013.

[65] L. Deri and N. SpA, "nProbe: an open source netflow probe for gigabit networks," in *TERENA Networking Conference*, 2003.

[66] C. M. Inacio and B. Trammell, "Yaf: yet another flowmeter," in *Proceedings of LISA'10: 24th Large Installation System Administration Conference*, p. 107, 2010.

[67] C. N. S. Suite, "Monitoring for Large-Scale Networks." [Online]. Available: `https://tools.netsa.cert.org/yaf/docs.html`, 2015. Accessed on 2018-07-07.

[68] paessler.com, "PRTG NETWORK MONITOR MANUAL." [Online]. Available: `https://www.paessler.com/manuals/prtg`. Accessed on 2018-06-24.

[69] K. Lakkaraju, W. Yurcik, R. Bearavolu, and A. J. Lee, "Nvisionip: an interactive network flow visualization tool for security," in *Systems, Man and Cybernetics, 2004 IEEE International Conference on*, vol. 3, pp. 2675–2680, IEEE, 2004.

[70] W. Yurcik, "VisFlowConnect-IP: a link-based visualization of netflows for security monitoring," in *18th Annual FIRST Conference on Computer Security Incident Handling*, pp. 1–10, 2006.

[71] Y. Li, A. Slagell, K. Luo, and W. Yurcik, "Canine: A combined conversion and anonymization tool for processing netflows for security," in *International conference on telecommunication systems modeling and analysis*, vol. 21, 2005.

[72] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss, "Quicksand: Quick summary and analysis of network data," tech. rep., Technical Report, Dec. 2001. citeseer. nj. nec. com/gilbert01quicksand. html, 2001.

[73] X. Li, F. Bian, M. Crovella, C. Diot, R. Govindan, G. Iannaccone, and A. Lakhina, "Detection and identification of network anomalies using sketch subspaces," in *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*, IMC '06, (New York, NY, USA), pp. 147–152, ACM, 2006.

[74] B. Krishnamurthy, S. Sen, Y. Zhang, and Y. Chen, "Sketch-based change detection: Methods, evaluation, and applications," in *Proceedings of the 3rd ACM SIGCOMM Conference on Internet Measurement*, IMC '03, (New York, NY, USA), pp. 234–247, ACM, 2003.

[75] G. Cormode and S. Muthukrishnan, "What's new: Finding significant differences in network data streams," *IEEE/ACM Trans. Netw.*, vol. 13, pp. 1219–1232, Dec. 2005.

[76] A. Soule, K. Salamatian, and N. Taft, "Combining filtering and statistical methods for anomaly detection," in *Proceedings of the 5th ACM SIGCOMM Conference on Internet Measurement*, IMC '05, (Berkeley, CA, USA), pp. 31–31, USENIX Association, 2005.

[77] S. Zhang, Y. Liu, D. Pei, Y. Chen, X. Qu, S. Tao, and Z. Zang, "Rapid and robust impact assessment of software changes in large internet-based services," in *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*, CoNEXT '15, (New York, NY, USA), pp. 2:1–2:13, ACM, 2015.

[78] M. Debashi and P. Vickers, "Sonification of network traffic flow for monitoring and situational awareness," *PloS one*, vol. 13, no. 4, p. e0195948, 2018.

[79] F. Silveira, C. Diot, N. Taft, and R. Govindan, "ASTUTE: detecting a different class of traffic anomalies," *SIGCOMM Comput. Commun. Rev.*, vol. 41, pp. –, Aug. 2010.

[80] A. Lakhina, M. Crovella, and C. Diot, "Diagnosing network-wide traffic anomalies," *SIGCOMM Comput. Commun. Rev.*, vol. 34, pp. 219–230, Aug. 2004.

[81] M. Cotton, L. Eggert, J. Touch, M. Westerlund, and S. Cheshire, "Internet assigned numbers authority (IANA) procedures for the management of the service name and transport protocol port number registry," BCP 165, RFC Editor, August 2011.

[82] M. Finsterbusch, C. Richter, E. Rocha, J.-A. Muller, and K. Hanssgen, "A survey of payload-based traffic classification approaches," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 2, pp. 1135–1156, 2014.

[83] T. T. Nguyen and G. Armitage, "A survey of techniques for internet traffic classification using machine learning," *IEEE Communications Surveys & Tutorials*, vol. 10, no. 4, pp. 56–76, 2008.

[84] D. Zuev and A. W. Moore, "Traffic classification using a statistical approach," in *International Workshop on Passive and Active Network Measurement*, pp. 321–324, Springer, 2005.

[85] M. Roesch *et al.*, "Snort: Lightweight intrusion detection for networks.," in *Lisa*, vol. 99, pp. 229–238, 1999.

[86] H. Kim, K. C. Claffy, M. Fomenkov, D. Barman, M. Faloutsos, and K. Lee, "Internet traffic classification demystified: myths, caveats, and the best practices," in *Proceedings of the 2008 ACM CoNEXT conference*, p. 11, ACM, 2008.

[87] H. Kim, M. Fomenkov, K. Claffy, N. Brownlee, D. Barman, and M. Faloutsos, "Comparison of internet traffic classification tools," *IMRG WACI*, vol. 2007, 2007.

[88] D. Rossi and S. Valenti, "Fine-grained traffic classification with netflow data," in *Proceedings of the 6th international wireless communications and mobile computing conference*, pp. 479–483, ACM, 2010.

[89] V. Carela-Espanol, P. Barlet-Ros, and J. Solé-Pareta, "Traffic classification with sampled netflow," *traffic*, vol. 33, pp. 34–34, 2009.

[90] J. Zhang, C. Chen, Y. Xiang, W. Zhou, and A. V. Vasilakos, "An effective network traffic classification method with unknown flow detection," *IEEE Transactions on Network and Service Management*, vol. 10, no. 2, pp. 133–147, 2013.

[91] P. Asrodia and H. Patel, "Analysis of various packet sniffing tools for network monitoring and analysis," *International Journal of Electrical, Electronics and Computer Engineering*, vol. 1, no. 1, pp. 55–58, 2012.

[92] S. Davidoff, *Network Forensics Tracking Hackers Through Cyberspace*, p. 95. England: Prentice Hall, 1st ed., 2012.

[93] E. S. Pilli, R. C. Joshi, and R. Niyogi, "Network forensic frameworks: Survey and research challenges," *digital investigation*, vol. 7, no. 1-2, pp. 14–27, 2010.

[94] A. Orebaugh, G. Ramirez, and J. Beale, *Wireshark & Ethereal network protocol analyzer toolkit*. Elsevier, 2006.

[95] P. Asrodia and H. Patel, "Network traffic analysis using packet sniffer," *International journal of engineering research and applications*, vol. 2, no. 3, pp. 854–856, 2012.

[96] QoSient, LLC, "Argus-auditing network activity." [Online]. Available: `https://qosient.com/argus/`. Accessed on 2018-06-24.

[97] S. Davidoff, *Network Forensics Tracking Hackers Through Cyberspace*, p. 163. England: Prentice Hall, 1st ed., 2012.

[98] CERT/NetSA at Carnegie Mellon University, "SiLK (System for Internet-Level Knowledge)." [Online]. Available: `http://tools.netsa.cert.org/silk`. Accessed on 2018-06-21.

[99] CERT/NetSA, "SiLK Documentation." [Online]. Available: `https://tools.netsa.cert.org/silk/docs.html`. Accessed on 2018-06-21.

[100] CERT Network Situational Awareness Group, "Using SiLK for network traffic analysis." [Online]. Available: `https://tools.netsa.cert.org/isilk/`. Accessed on 2018-06-24.

[101] A. Abdalla, H. A. Jamil, H. IBRAHIM, H. AWAD, and S. M. NOR, "Malware dection using ip flow level attributes.," *Journal of Theoretical & Applied Information Technology*, vol. 57, no. 3, 2013.

[102] A. Gouveia and M. Correia, "Feature set tuning in statistical learning network intrusion detection," in *15th IEEE International Symposium on Network Computing and Applications*, pp. 68–75, 2016.

[103] L. Sacramento, I. Medeiros, J. Bota, and M. Correia, "Flowhacker: Detecting unknown network attacks in big traffic data using network flows," in *Proceedings of IEEE Trustcom*, 2018.

[104] I. Sommerville, *Software Engineering*. England: Addison-Wesley, 1998.

[105] S. Pfleeger, *Software Engineering Theory and Practice*, ch. 4, pp. 139–142. United States of America: Prentice Hall, 2001.

[106] P. Clements, D. Garlan, R. Little, R. Nord, and J. Stafford, "Documenting software architectures: views and beyond," in *Proceedings of the 25th International Conference on Software Engineering*, pp. 740–741, IEEE Computer Society, 2003.

[107] P. C. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord, and J. Stafford, "A practical method for documenting software architectures," 2002.

[108] I. Jacobson, "Object-oriented development in an industrial environment," *SIGPLAN Not.*, vol. 22, pp. 183–191, Dec. 1987.

[109] G. L. Fenves, “Object-oriented programming for engineering software development,” *Engineering with computers*, vol. 6, no. 1, pp. 1–15, 1990.

[110] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin, “Aspect-oriented programming,” in *European conference on object-oriented programming*, pp. 220–242, Springer, 1997.

[111] M. Fowler, *UML distilled: a brief guide to the standard object modeling language*. Addison-Wesley Professional, 2004.

[112] B. Selic, “Using UML for modeling complex real-time systems,” in *Languages, compilers, and tools for embedded systems*, pp. 250–260, Springer, 1998.

[113] D. Berardi, D. Calvanese, and G. De Giacomo, “Reasoning on UML class diagrams,” *Artificial intelligence*, vol. 168, no. 1-2, pp. 70–118, 2005.

[114] Schalk, C., “JavaServer Faces Technology.” [Online]. Available: `http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html`. Accessed on 2018-06-18.

[115] A. Leff and J. T. Rayfield, “Web-application development using the model/view/controller design pattern,” in *Enterprise Distributed Object Computing Conference, 2001. EDOC’01. Proceedings. Fifth IEEE International*, pp. 118–127, IEEE, 2001.

[116] D. Schwabe and G. Rossi, “An object oriented approach to web-based applications design,” *TAPOS*, vol. 4, no. 4, pp. 207–225, 1998.

[117] C. Çivici, *PrimeFaces User Guide*. primefaces.org, https://www.primefaces.org. Accessed on 2018-06-18.

[118] C. Pautasso, E. Wilde, and A. Marinos, “First international workshop on restful design (ws-rest 2010),” in *Proceedings of the First International Workshop on RESTful Design*, WS-REST ’10, (New York, NY, USA), pp. 1–3, ACM, 2010.

[119] J. Strauch and S. Schreier, “Restify: From rpcs to restful http design,” in *Proceedings of the Third International Workshop on RESTful Design*, WS-REST ’12, (New York, NY, USA), pp. 11–18, ACM, 2012.

[120] Google Inc., https://developers.google.com/maps/documentation/, *Google Maps Platform Documentation*. Accessed on 2018-06-21.

[121] R. Hofstede and T. Fioreze, “Surfmap: A network monitoring tool based on the google maps api,” in *Integrated Network Management, 2009. IM’09. IFIP/IEEE International Symposium on*, pp. 676–690, IEEE, 2009.

[122] Neutrino, https://www.neutrinoapi.com/api/api-basics/, *API Fundamentals*. Accessed on 2018-06-21.

[123] Cmap, “Documentation & Support.” [Online]. Available: `https://cmap.ihmc.us/documentation-support/`. Accessed on 2018-07-10.

[124] Neutrino, https://www.neutrinoapi.com/api/ip-blocklist/, *IP Blocklist*. Accessed on 2018-06-21.

[125] ip-api.com, http://ip-api.com/docs/, *IP Geo Location*. Accessed on 2018-06-21.

[126] ip-api.com, http://ip-api.com/docs/api:json, *JSON*. Accessed on 2018-06-21.

[127] ubuntu.com, https://wiki.ubuntu.com/TrustyTahr/ReleaseNotes/UbuntuGNOME, *Ubuntu GNOME 14.04 LTS*. Accessed on 2018-06-21.

[128] netsa.cert.org, https://tools.netsa.cert.org/confluence/pages/viewpage.action?pageId=23298051, *SiLK on a Box Ubuntu 12.04 Standalone Flow Collection and Analysis*. Accessed on 2018-06-21.

[129] netsa.cert.org, https://tools.netsa.cert.org/silk/silk-install-handbook.html, *SiLK Installation Handbook SiLK-3.17.0*. Accessed on 2018-06-21.

[130] SEI CMU, https://www.youtube.com/user/TheSEICMU/playlists, *How to Install NetSA Tools on Ubuntu*. Accessed on 2018-06-21.

[131] Netbeans.org, "What's the Difference between NetBeans Platform and Eclipse RCP?." [Online]. Available: `https://netbeans.org/features/platform/compare.html`. Accessed on 2018-06-22.

[132] D. C. Schmidt, M. Stal, H. Rohnert, and F. Buschmann, *Pattern-Oriented Software Architecture, Patterns for Concurrent and Networked Objects*, vol. 2. John Wiley & Sons, 2013.

[133] J. Hannemann and G. Kiczales, "Design pattern implementation in java and AspectJ," in *ACM Sigplan Notices*, vol. 37, pp. 161–173, ACM, 2002.

[134] R. C. Martin, *Agile software development: principles, patterns, and practices*. Prentice Hall, 2002.

[135] netsa.cert.org, https://tools.netsa.cert.org/silk/referencedata.html, *Silk Reference Data*. Accessed on 2018-06-18.

[136] A. D. Kent, "Comprehensive, Multi-Source Cyber-Security Events." Los Alamos National Laboratory, 2015.

[137] A. D. Kent, "Cyber security data sources for dynamic network research," in *Dynamic Networks and Cyber-Security*, pp. 37–65, World Scientific, 2016.

Appendix A

Neutrino API

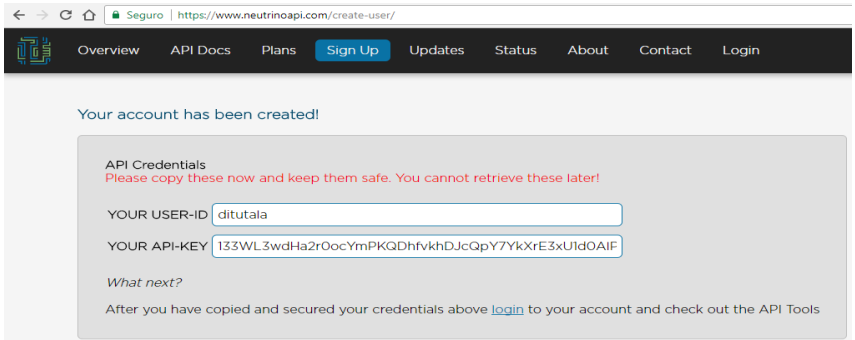


Figure A.1: Neutrino API Account

API Response

- JSON
- XML

Parameter	Type	Description
ip	string	The IP address
is-listed	boolean	Is this IP on a blocklist
list-count	integer	The number of blocklists the IP is listed on
blocklists	array	An array of strings indicating which blocklists this IP is listed on (empty if not listed)
last-seen	integer	The last time this IP was seen on a blocklist (in Unix time or 0 if not listed recently)
is-proxy	boolean	IP has been detected as an anonymous web proxy or anonymous HTTP proxy
is-tor	boolean	IP is a Tor node or running a Tor related service
is-vpn	boolean	IP has been detected as belonging to a VPN provider
is-malware	boolean	IP is involved in distributing or is running malware
is-spyware	boolean	IP is involved in distributing or is running spyware
is-dshield	boolean	IP has been flagged as an attack source on DShield (dshield.org)
is-hijacked	boolean	IP is part of a hijacked netblock or a netblock controlled by a criminal organization
is-spider	boolean	IP is running a hostile web spider / web crawler
is-bot	boolean	IP is hosting a malicious bot or is part of a botnet. Includes brute-force crackers
is-spam-bot	boolean	IP address is hosting a spam bot, comment spamming or any other spamming type software
is-exploit-bot	boolean	IP is hosting an exploit finding bot or is running exploit scanning software

Figure A.2: Neutrino API IP BlockList Output

Appendix B

IPDiff Outputs

The Figure B.1 shows how IPDiff classes were divided into packages.

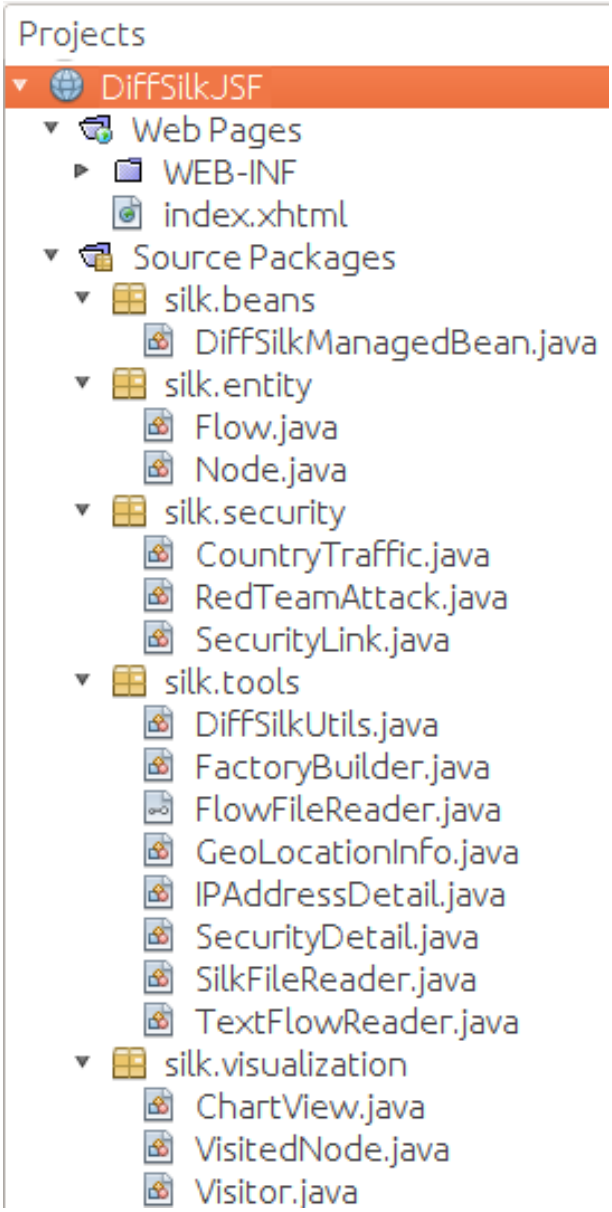


Figure B.1: IPDiff Packages

If all installation process has sucessed, when the application is run under the NetBeans platform, it is possible to see the url to run in the web browser, as in Figure B.2

```
Output
DiffSilkJSF (run) ×   Java DB Database Process ×   GlassFish Server 4.1.1 ×
ant -f /home/ditutala/DiffSilkJSF -Dnb.internal.action.name=run -Dc
t.for.caches=true -Dbrowser.context=/home/ditutala/DiffSilkJSF run
init:
deps-module-jar:
deps-ear-jar:
deps-jar:
library-inclusion-in-archive:
library-inclusion-in-manifest:
compile:
compile-jsps:
Starting GlassFish Server 4.1.1
GlassFish Server 4.1.1 is running.
Incrementally deploying DiffSilkJSF
Completed incremental distribution of DiffSilkJSF
run-deploy:
Browsing: http://localhost:8080/DiffSilkJSF
run-display-browser:
run:
BUILD SUCCESSFUL (total time: 3 minutes 45 seconds)
```

Figure B.2: IPDiff GlassFish Run Successfully

The first layout used for IPDiff interface can be seen in Figure B.3. It was abandoned because in many cases where a huge number of flows found the chart information were not clear for reading.

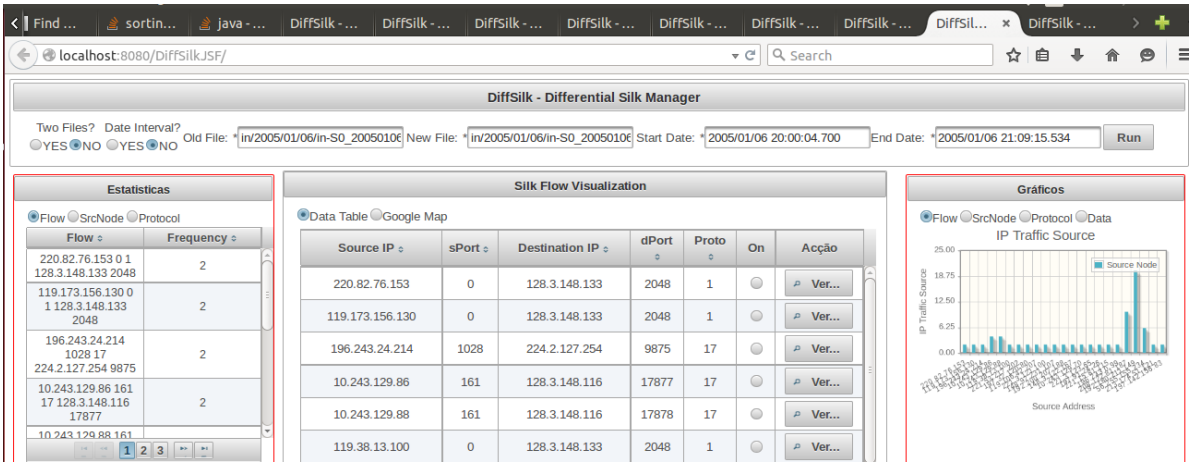


Figure B.3: First Layout Used

Figure B.4 shows the selected flow source node IP address details.

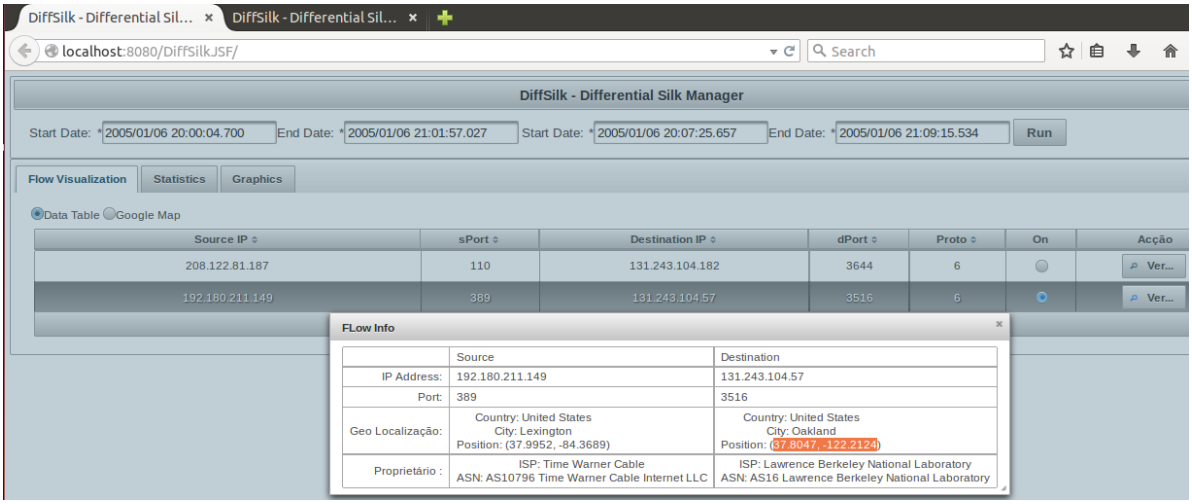


Figure B.4: Source Node Detailed Information

Appendix C

IPDiff Experimental

Due to memory limitation the LANL dataset flow file of about 5GB was split into smaller files, according to the experiment. Figure C.1 shows the output of ls -lh unix command, executed under the working directory.

```
ditutala@ditutala-VirtualBox:/media/UBUNTU-SILK/LANL DATA SET/5_Milhoes_Linha$ split -l 400000 -d --additional-suffix=.txt flowFile15.txt flowFi
le15
ditutala@ditutala-VirtualBox:/media/UBUNTU-SILK/LANL DATA SET/5_Milhoes_Linha$ split -l 400000 -d --additional-suffix=.txt flowFile16.txt flowFi
le16
ditutala@ditutala-VirtualBox:/media/UBUNTU-SILK/LANL DATA SET/5_Milhoes_Linha$ split -l 400000 -d --additional-suffix=.txt flowFile17.txt flowFi
le17
ditutala@ditutala-VirtualBox:/media/UBUNTU-SILK/LANL DATA SET/5_Milhoes_Linha$ split -l 400000 -d --additional-suffix=.txt flowFile18.txt flowFi
le18
ditutala@ditutala-VirtualBox:/media/UBUNTU-SILK/LANL DATA SET/5_Milhoes_Linha$ split -l 400000 -d --additional-suffix=.txt flowFile19.txt flowFi
le19
ditutala@ditutala-VirtualBox:/media/UBUNTU-SILK/LANL DATA SET/5_Milhoes_Linha$ split -l 400000 -d --additional-suffix=.txt flowFile20.txt flowFi
le20
ditutala@ditutala-VirtualBox:/media/UBUNTU-SILK/LANL DATA SET/5_Milhoes_Linha$ split -l 400000 -d --additional-suffix=.txt flowFile21.txt flowFi
le21
ditutala@ditutala-VirtualBox:/media/UBUNTU-SILK/LANL DATA SET/5_Milhoes_Linha$ split -l 400000 -d --additional-suffix=.txt flowFile22.txt flowFi
le22
ditutala@ditutala-VirtualBox:/media/UBUNTU-SILK/LANL DATA SET/5_Milhoes_Linha$ split -l 400000 -d --additional-suffix=.txt flowFile23.txt flowFi
le23
```

Figure C.1: LANL dataset flow file splited

Figure C.2 shows redTeam flows from a redTeam computer (in blue) with several destinations (in green). This may suggest a redTeam user trying to scan vulnerable targets.

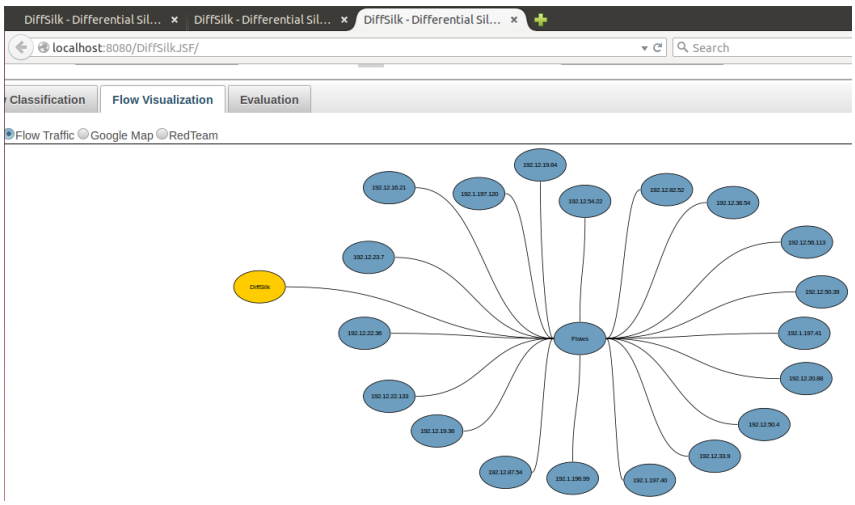


Figure C.2: RedTeam flows between a source (in blue) and the target computers (in green)

Appendix D

IPDiff Code

```
public static Date convertIntTimeDate(String start, String strDuration)
           throws ParseException {
  int startYear = 2015;
  Calendar calendar = Calendar.getInstance(TimeZone.getTimeZone("UTC"));
  calendar.clear();
  calendar.set(startYear, Calendar.JANUARY, 1);
  long secondsSinceEpoch = calendar.getTimeInMillis();

  long startTime = Long.parseLong(start.trim());
  // +1s to differentiate sTime to eTime on strDuration = "0"
  long duration = Long.parseLong(strDuration.trim()) + 1;
  long endTime = startTime + duration;

  String sdate;
  sdate = new SimpleDateFormat("yyyy/mm/dd␣HH:mm:ss")
  .format(new Date(endTime * 1000 + secondsSinceEpoch));
   SimpleDateFormat date;
   date = new SimpleDateFormat("yyyy/mm/dd␣HH:mm:ss").parse(sdate)
  return date;
}
```

Listing D.1: String to Date Conversion