IPDiff - Detecting IP Traffic Changes

Mbuku Tunga Ditutala
Instituto Superior Técnico, Universidade de Lisboa

***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.**

I. 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.

A. 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 occurs.

B. 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 fails 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.

C. 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? Are there traffic from this or that protocol? Is there an attack? and many more. To answer to 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 analyse 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.

D. Thesis Outline

The remaining of this paper can be summarised as follow: Section 2 describes the main architectural design decisions and frameworks for web application development. It also presents IPDiff implementation and deployment environment. Section 3 presents the IPDiff evaluation and results discussion. Finally in Section 4 main findings and conclusions are presented.

II. IPDiff Design and Implementation

This chapter provides an overview of the architecture of IPDiff and describes its implementations. 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.

A. IPDiff Architectural Design

This section describes the design decisions taken when developing IPDiff.

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, Border Gateway Protocol (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 1 shows the IPDiff system context diagram.

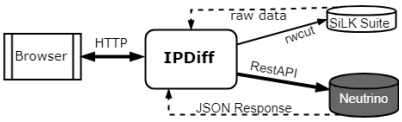

Fig. 1: IPDiff Context Diagram

2) Requirements: A requirement can be a function or a service that the system can do and its operational environment constraints [21]. 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 [22]. (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. [21].

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) 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 [23], [24].

In object oriented programming, system elements can be represented as objects that behave as those elements [25]–[27]. According to the list of requirements presented in Section II-A2, 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 [21], [23]. The IPDiff main components are shown in Figure 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.

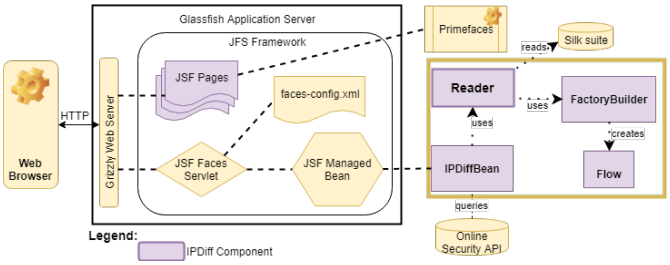

Fig. 2: IPDiff Decomposition Style

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 [21], [28]–[30]. The IPDiff class diagram can be seen in Figure 3. A short description of the IPDiff main classes comes below:

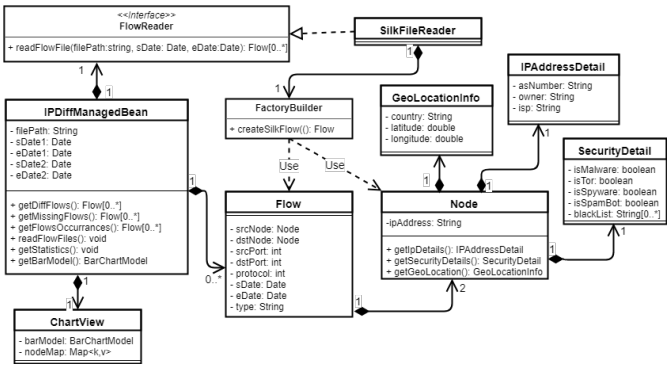

Fig. 3: IPDiff Class Diagram

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 FlowRader in turn delegates the flow creation process to a **FactoryBuilder** object, which creates first each of the two **Node** and finally the **Flow**. The flow created are them 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.

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 [31].

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 handle 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 [32], [33]. 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 [31].

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 the 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 has 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 [34].

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 [35]. The Representational State Transfer (REST) a resource-oriented architectural style is widely used by resource aware applications [36]. 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 [37]. 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 feature 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 [38].

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. [39].

Mind map software are 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 the concept of mind map, such as, the FreeMind widely used for expressing brainstorming session in a visual diagram, and Cmap [40].

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.

B. Flow Processing

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

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 read the file in SiLK format and to process its output line by line to build the flows.

2) Flow Classification: Filtering flows to be analysed is similar to the approach used in [41]. 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 has flows already analysed or old flow, 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 that it belongs to that interval. In Figure 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 that do not belong to that interval while the others flows belongs, 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) 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

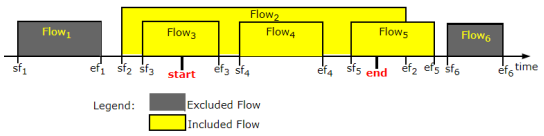

Fig. 4: Flow Processing Time Interval

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

the internet. IPDiff rely on Neutrino API, to gather security information related to an IP address [39]. 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 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, a Exploit scanners [42].

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 [43]. 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, Autonomous System (AS) number, as shown in Figure 5, in case of successful query [44]. 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"
}
```

Fig. 5: IP-API JSON output parameters

C. IPDiff Deployment

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

1) Deployment Environment: 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 [45].

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 [46], [47]. I also recommend to check NetSA install tools play list videos on youtube [48]. The following two images demonstrate a successfully SiLK suite installation 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 6.

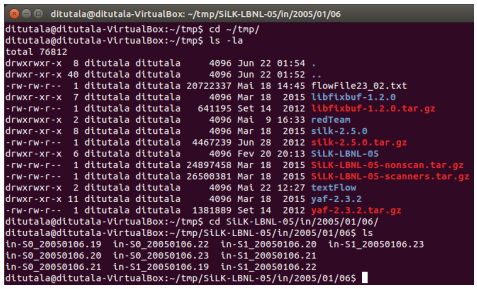

Fig. 6: SiLK Directory and File Structure and Naming

The Figure 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|
```

Fig. 7: SiLK rwcut Command Output

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 the case for IPDiff, for IPDiff development NetBeans IDE 8.2 was chosen because of its built in tools [49]. The GlassFish 4.1.1 is part of those tools.

4) IPDiff Layout Interface: The 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 [50]–[52].

D. 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'. 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 data.
- Old flows end date, a string representing a data.
- New flows start date, a string representing a data.
- New flows end date, a string representing a data.

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 8. 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.

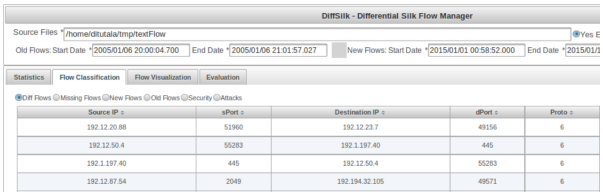

Fig. 8: IPDiff Flow Classification

Figure 9 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 II-B3.

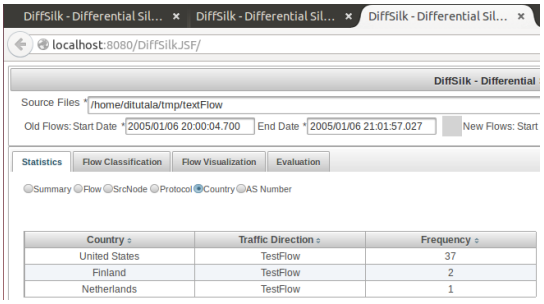

Fig. 9: IPDiff - Per Country Statistics

Another output produced by IPDiff is the visualization of traffic between nodes. The MindMap Primeface UI component was the choice. The Figure 10 shows network flow end points, that is, flows from the source node (blue in the middle) to several destination node (in green). Each node is identified by its IP address.

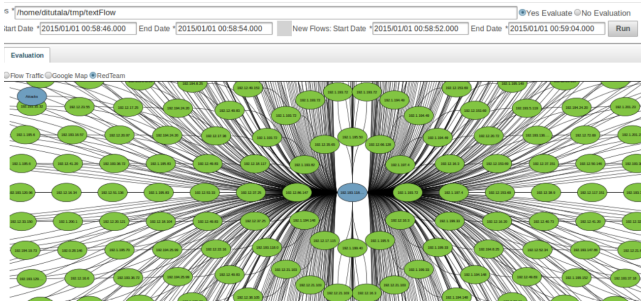

Fig. 10: IPDiff Visualization of Node Traffic

The last image, in Figure 11 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 Netherlands, which confirms what was shown in the country statistics output, in Figure 9.

E. Summary

In this chapter, a new tool for network change detection has been presented, which compare flows based on time intervals.

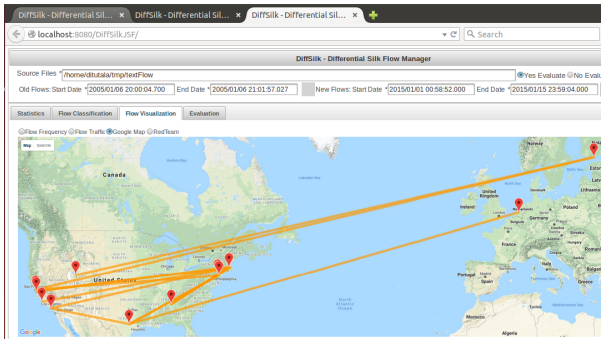

Fig. 11: IPDiff Traffic Visualization on Google Map

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 a was used.

III. Evaluation

This chapter describes the evaluation process of IPDiff and discuss 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.

A. Evaluation Dataset

To evaluate the ability of IPDiff to process flow in different format, 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 [53]. 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) [54] 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 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 [55].

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 [55], as can be seen in Figure 12.

```
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

Fig. 12: LANL Dataset Files Used

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 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 1: IP Address Padding and Conversion

A similar procedure was used to determine a flow start and end time. 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.

B. 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. The data used for all experiments can be seen in the Figure 13.

#	File	FileSize[MB]	start2	end2	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 09:02:45.000	2015/01/13 10:01:30.000	0	32	0	0	0	0
14	flowFile20.txt	32	2015/01/13 09:35:00.000	2015/01/13 09:50:30.000	102,643	32	99127	105933	38373	32756
15	flowFile20.txt	32	2015/01/13 09:30:00.000	2015/01/13 10:00:30.000	78,448	32	99127	208822	78932	14583
16	flowFile21.txt	48	2015/01/13 11:50:00.000	2015/01/13 12:30:00.000	110,236	48	30063	243426	183033	0
17	flowFile2207.txt	16	2015/01/14 17:00:00.000	2015/01/14 17:48:45.000	17,464	16	158996	208523	205097	156944
18	flowFile2207.txt	16	2015/01/14 17:30:00.000	2015/01/14 17:51:00.000	169,02	16	108061	81188	79730	106708

Fig. 13: Experimental data

IPDiff was developed with the aim of detecting any new and unknown network flow based on time interval. In total, 18 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 14. The output in Figure 14a 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 14b 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.

Another aim of IPDiff was to find out detailed security information about a particular flow that belongs to the diff flows. The Figure 15 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	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

Fig. 14: Experimental Results from two runs with two files

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

Fig. 15: Diff flows and Missing Flows Detected

The IPDiff performance, in a different perspective, can be seen in Figure 16. 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.

The Figure 17 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

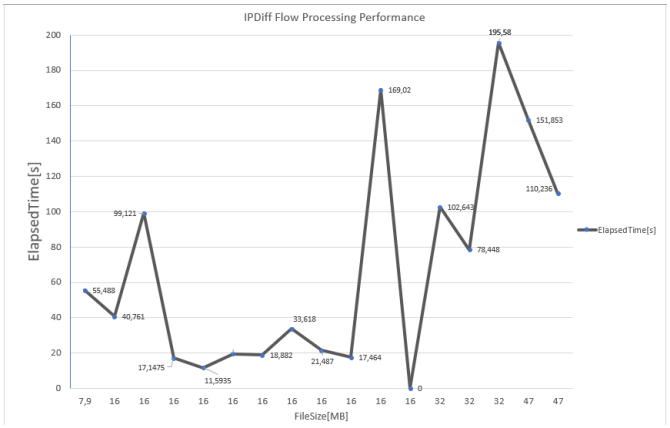

Fig. 16: IPDiff Performance Evaluation

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 to the number of missing flows in all cases where the number of new flows is greater to the number of old flows, but this does not hold in 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 13), 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.

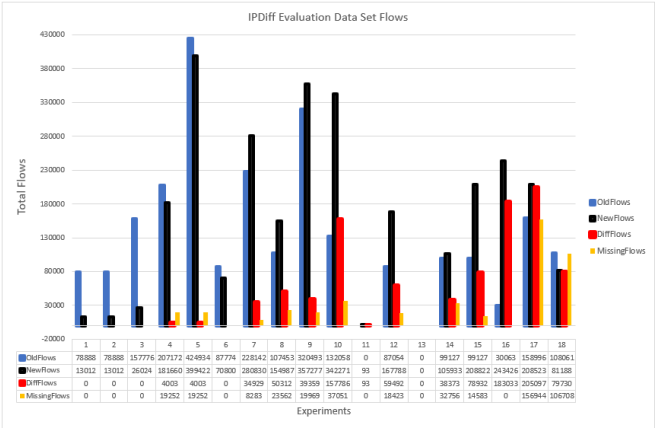

Fig. 17: Experimental Results Summary

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 18.

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

Fig. 18: Maximum diff flows processed

C. 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 17.

Although machine learning supervised or unsupervised techniques were not used in IPDiff as in [56], Figure 17 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.

IV. 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 occurs. IPDiff experimental results suggest that by using network flows network management tools are capable of answering those questions.

The IPDiff experimental results adds to our understanding of network flow processing and how it can be used for network traffic change detection and classification. One of the more significant findings to emerge from this study is that when processing huge flow files, there are much computation time depending on amount of interesting flows found on the file.

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.

References

[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 2018-08-07.

[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, 1998, pp. 141–148.

[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*. IEEE, 2011, pp. 19–24.

[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*. IEEE, 2009, pp. 186–198.

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

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

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

[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*, 2010, p. 119.

[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, no. 4. ACM, 2010, pp. 303–314.

[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*. ACM, 2011, p. 13.

[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*, ser. Workshops ICDCN '18. New York, NY, USA: ACM, 2018, pp. 2:1–2:6. [Online]. Available: http://doi.acm.org/10.1145/3170521.3170523

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

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

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

[23] 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*. IEEE Computer Society, 2003, pp. 740–741.

[24] 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.

[25] I. Jacobson, "Object-oriented development in an industrial environment," *SIGPLAN Not.*, vol. 22, no. 12, pp. 183–191, Dec. 1987. [Online]. Available: http://doi.acm.org/10.1145/38807.38824

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

[27] 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*. Springer, 1997, pp. 220–242.

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

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

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

[31] C. Schalk, *JavaServer Faces Technology*, oracle, http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html, accessed on 2018-06-18. [Online]. Available: http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html

[32] 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*. IEEE, 2001, pp. 118–127.

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

[34] C. Çivici, *PrimeFaces User Guide*, primefaces.org, https://www.primefaces.org, accessed on 2018-06-18. [Online]. Available: https://www.primefaces.org

[35] 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*, ser. WS-REST '10. New York, NY, USA: ACM, 2010, pp. 1–3. [Online]. Available: http://doi.acm.org/10.1145/1798354.1798375

[36] J. Strauch and S. Schreier, "Restify: From rpcs to restful http design," in *Proceedings of the Third International Workshop on RESTful Design*, ser. WS-REST '12. New York, NY, USA: ACM, 2012, pp. 11–18. [Online]. Available: http://doi.acm.org/10.1145/2307819.2307824

[37] *Google Maps Platform Documentation*, Google Inc., https://developers.google.com/maps/documentation/, accessed on 2018-06-21. [Online]. Available: https://developers.google.com/maps/documentation/

[38] 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*. IEEE, 2009, pp. 676–690.

[39] *API Fundamentals*, Neutrino, https://www.neutrinoapi.com/api/api-basics/, accessed on 2018-06-21. [Online]. Available: https://www.neutrinoapi.com/api/api-basics/

[40] Cmap, "Documentation & Support," [Online]. Available: https://cmap.ihmc.us/documentation-support/., accessed on 2018-07-10.

[41] 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*, ser. IMC '05. Berkeley, CA, USA: USENIX Association, 2005, pp. 31–31. [Online]. Available: http://dl.acm.org/citation.cfm?id=1251086.1251117

[42] *IP Blocklist*, Neutrino, https://www.neutrinoapi.com/api/ip-blocklist/, accessed on 2018-06-21. [Online]. Available: https://www.neutrinoapi.com/api/ip-blocklist/

[43] *IP Geo Location*, ip-api.com, http://ip-api.com/docs/, accessed on 2018-06-21. [Online]. Available: http://ip-api.com/docs/

[44] *JSON*, ip-api.com, http://ip-api.com/docs/api:json, accessed on 2018-06-21. [Online]. Available: http://ip-api.com/docs/api:json

[45] *Ubuntu GNOME 14.04 LTS*, ubuntu.com, https://wiki.ubuntu.com/TrustyTahr/ReleaseNotes/UbuntuGNOME, accessed on 2018-06-21. [Online]. Available: https://wiki.ubuntu.com/TrustyTahr/ReleaseNotes/UbuntuGNOME

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

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

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

[49] 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.

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

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

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

[53] *Silk Reference Data*, netsa.cert.org, https://tools.netsa.cert.org/silk/referencedata.html, accessed on 2018-06-18. [Online]. Available: https://tools.netsa.cert.org/silk/referencedata.html

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

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

[56] 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.