Comparison of Intrusion Detection Systems for Use in Low-Powered Devices

A project report submitted

to

MANIPAL ACADEMY OF HIGHER EDUCATION

For Partial Fulfillment of the Requirement for the

Award of the Degree

of

Bachelor of Technology

in

Computer and Communication Engineering

by

Pratyay Amrit

Reg. No. 140953430

Under the guidance of

Ms. Ipsita Upasana
Assistant Professor
Department of I & CT
Manipal Institute of Technology
Manipal, India

MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

INSPIRED BY LIFE

MAY 2018

I dedicate my thesis to my friends and family.

DECLARATION

I hereby declare that this project work entitled **Comparison of Intrusion Detection Systems for Use in Low-Powered Devices** is original and has been carried out by me in the Department of Information and Communication Technology of Manipal Institute of Technology, Manipal, under the guidance of **Ms. Ipsita Upasana**, **Assistant Professor**, Department of Information and Communication Technology, M. I. T., Manipal. No part of this work has been submitted for the award of a degree or diploma either to this University or to any other Universities.

Place: Manipal

Date : 05-05-18

Pratyay Amrit

MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

CERTIFICATE

This is to certify that this project entitled **Comparison of Intrusion Detection Systems for Use in Low-Powered Devices** is a bonafide project work done by **Mr. Pratyay Amrit (Reg. No.: 140953430)** at Manipal Institute of Technology, Manipal, independently under my guidance and supervision for the award of the Degree of Bachelor of Technology in Computer and Communication Engineering.

Ms. Ipsita Upasana	Dr.Balachandra
Assistant Professor	Professor & Head
Department of I & CT	Department of I & CT
Manipal Institute of Technology	Manipal Institute of Technology
Manipal, India	Manipal, India

ACKNOWLEDGEMENTS

My sincere thanks to Ms. Ipsita Upasana, and all faculty members of the Department of Information & Communcation Technology for providing necessary guidance and feedback throughout the course of the project.

ABSTRACT

Intrusion Detection Systems have become an essential part of computer network security. It acts as a first-line defense from cyber-attacks by analyzing the various attributes of a data packet and identifying it as a malicious, or an ordinary one. Such systems have come a long way and in the present time, promise acceptable performance. However, the algorithms that run at the core of these systems are computationally expensive, making them fairly accurate, but almost unimplementable in very low powered devices, such as nodes of a WSAN, or in an IoT based setup. This work attempts to rank various machine learning and data mining techniques on the basis of their accuracy and time required to train and classify network data.

[**Security and Privacy**]: Intrusion/Anomaly Detection and Malware Mitigation–Intrusion detection systems

Contents

List of Tables

List of Figures

ABBREVIATIONS

IDS	:	Intrusion Detection System
UNSW-NB15	:	University of New South Wales, Network Based 2015 Dataset
KDD'99	:	Knowledge Discovery and Data Mining Competition 1999 Dataset
SIEM	:	System Information and Event Management
API	:	Application Programming Interface

NOTATIONS

α : Smoothing factor for words

β : Smoothing factor for topics

Chapter 1

Introduction

1.1 Background

An Intrusion Detection System (IDS) is an application that can be installed on a system or a network. The application may be a piece of software, or a device that can be plugged into the interface as a module. The IDS, once plugged into the system, scans for activity, and based on various features pertaining to the activity, classifies it as normal, or malicious. It must be noted that the IDS itself does not provide any access control, and is only responsible for the *detection* of an attack. As a result, IDSs must be bundled with other tools to prevent an attack. Typically, all detected malicious activity is reported to a Security Information and Event Management (SIEM) system, which combines reports from multiple sources and uses alarm filtering techniques to distinguish malicious activity from false alarms.

Many attempts have been made to classify various kinds of IDSs. According to [1], on the basis of method of detection, two types of IDSs have been identified:

- **Knowledge Based**: Knowledge Based IDSs accumulate knowledge about attacks and look for similar data to detect an attack. Since these IDSs look for very specific signatures, they provide very accurate detection of common attacks. However, in a case where an attack has a signature that has never been seen before, this kind of system fails. This type of IDS is also called **misuse detection** IDS.

- **Behavior Based**: Behavior Based IDSs work with the assumption that attacks can be detected if the behavior of the system or the users deviate from the normal expected pattern. Many sources are combined to gather enough data to teach a system what normal behavior looks like and to create a model. The IDS then compares the current activity with this model and reports if it detects an anomaly. These IDSs are also called **anomaly detection** IDSs. Since these IDSs are based on predictive models, they tend to have lower accuracy, and higher false alarm rate when compared to a knowledge based IDS. However, they are effective against novel attacks as any kind of attack is detected as an anomaly, or a deviation from the norm.

A third type of IDS is also often seen in literature, called the **hybrid detection** IDS. This is a combination of the two types. Typically, an activity is first passed through a knowledge based system. If the first filter claims the activity to be an attack, the system terminates and the activity is reported. If the activity passes the first filter, it is passed through a behavior based system, where even if the signature could not be found in the dictionary of attacks, an anomaly is detected and the activity is reported. The signature of this activity is then recorded in the knowledge based system for future reference. An activity is not reported as malicious only if it passes both of the filters. Although the process of identification is often sped up compared to the other two types of IDSs, the accuracy is bottle-necked by the behavior based system used. Running such a system with a poor behavior based system also possess the risk of saving false signatures in the initial filter. If an activity passes through the first filter, and the second filter raises a false alarm, the signature of the (in reality) normal activity is recorded as malicious, and is reported in subsequent instances of similar activities.

Thus, choosing the right behavior based model becomes an important part of building an IDS. Knowledge based systems are not future proof, and it is often too late for a system or a network to be affected by an attacker even once, regardless of how well the IDS reports subsequent activities of the same. Hybrid systems may provide quick classification, but are bottle-necked by the high false positive rate of the behavior based model used.

1.2 Objectives

The broad objective of this study was to compare different machine learning and data mining models on the basis of their accuracy as well as execution time.
Specifically, the objectives can be broken down to the following:

- To determine useful features in classifying network data.
- To find appropriate measures to score the different models.
- To study the causes of different models performing differently.
- To open scope for future studies to improve the given score.

1.3 Limitations of Study

- The study has been conducted with a static database for training and testing. Although the dataset has been curated fairly recently (2015), it is at best, a rough approximate of live data flowing through a network.

- Many complex algorithms (such as deep neural networks, multi-layer perceptron etc.) which have proved to be very accurate have not been tested simply because the execution time of these algorithms were too high for it to fetch a comparable score.

- Although [2] suggests a faster and efficient way to implement a multi-layer perceptron with binary weights, it has also claimed a reduction of accuracy by up to 4 times for the same, and hence has not been considered in the comparison.

Chapter 2

Literature Review

2.1 Taxonomy for Intrusion Detection Systems

In [1], an attempt has been made to standardize a terminology for IDSs. Many different types of common in use IDSs are introduced in the paper, along with some upcoming possibilities.

Broadly, IDSs have been classified on the basis of 5 different features (figure 2.1).

1. On the basis of detection method:
    - Behavior based
    - Knwoledge based
2. On the basis of behavior on detection:
    - Passive filtering
    - Active filtering
3. On the basis of audit source location:
    - Host log files
    - Network packets

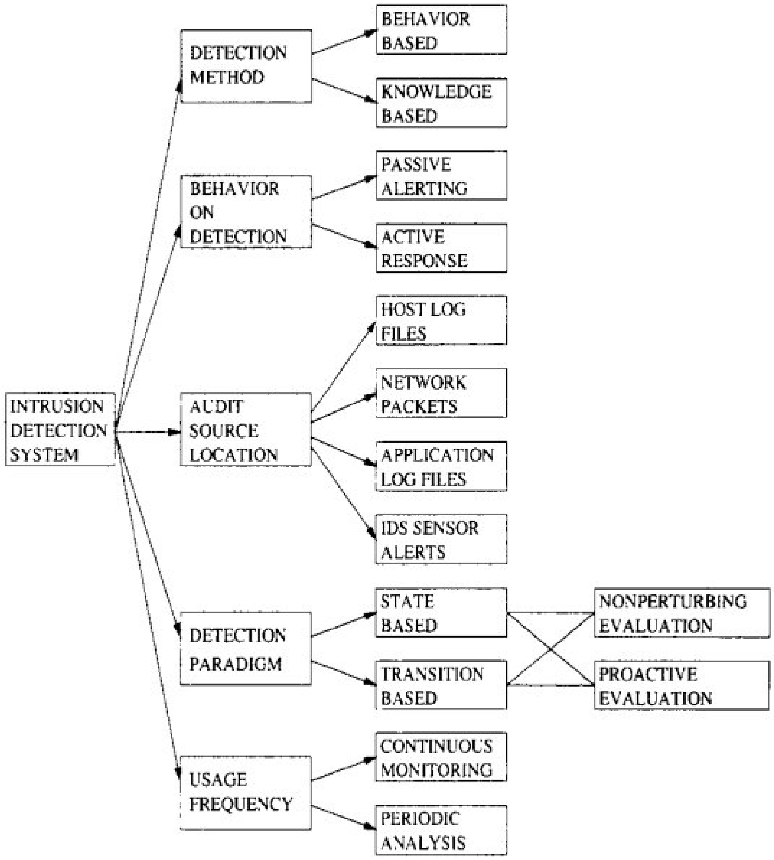

Figure 2.1: Types of IDS as mentioned in [1]

- Application log files
- IDS sensor alerts

4. On the basis of detection paradigm:
   - State based
   - Transition based

5. On the basis of usage frequency:
   - Continuous monitoring
   - Periodic analysis

Chapter 3

Chapter Title

3.1 ffffh

Chapter 4

Chapter Title

4.1 abvvv

Chapter 5

Chapter Title

5.1 ghf

Chapter 6

Chapter Title

6.1 vvvvvv

Chapter 7

Conclusion

7.1 fff

Appendices

Appendix A

Code (if required)

A.1 Kerberos Protocol

```
MODULE main

VAR

--Creating agents which are to type agtype
agA : agtype;
agB : agtype;
agS : agtype;
agI : agtype;
Iactive: boolean;

--Assigning initial to values to all variables

ASSIGN
init(agA.state):=wait;
init(agB.state):=wait;
init(agS.state):=wait;
init(agI.state):=wait;
init(agA.count):=0;
init(agB.count):=0;
init(agS.count):=0;
init(agI.count):=0;
init(agA.authenticated):=FALSE;
init(agB.authenticated):=FALSE;
init(agI.authenticated):=FALSE;
init(agS.authenticated):=TRUE;
```

```
--Transitions for the variable indicating presence or absence of intruder

next(Iactive):=
case
!Iactive:{0,1};
Iactive & agI.state=receive4beta:{0,1};
1:Iactive;
esac;

--Transitions for agent A's state

next(agA.state):=
case
agA.state=wait: send1;
agS.state=send2 & agA.state=send1: receive2;
agA.state=receive2: send3alpha;
agB.state=send4alpha & agA.state=send3alpha: receive4alpha;
agA.state=receive4alpha: wait;
1:agA.state;
esac;

--Transitions for agent B's state

next(agB.state):=
case
agA.state=send3alpha & agB.state=wait: receive3alpha;
agI.state=send3beta & agB.state=wait & Iactive: receive3beta;
agI.state=send3beta & agB.state=send4alpha & Iactive: receive3beta;
agB.state=receive3alpha:send4alpha;
agB.state=receive3beta:send4beta;
agB.state=send4alpha:wait;
1:agB.state;
esac;

--Transitions for Server S's state

next(agS.state):=
case
agA.state=send1 & agS.state=wait: receive1;
agS.state=receive1:send2;
agS.state=send2:wait;
1:agS.state;
esac;
```

```
--Transitions for the Intruder's state

next(agI.state):=
case
agI.state=wait & agA.state=send3alpha & agB.state=wait & Iactive: receive3beta;
agI.state=receive3beta & Iactive: send3beta;
agI.state=send3beta & agB.state=send4beta & Iactive  : receive4beta;
agI.state=receive4beta & Iactive: wait;
1:agI.state;
esac;

--Transitions for Agent A's counter

next(agA.count):=
case
agA.state=send1|agA.state=receive2: agA.count;
agA.state=send3alpha & agA.count<1:agA.count+1;
agA.count=1 & agA.state=receive2: 0;
1:agA.count;
esac;

--Transitions for Agent B's counter

next(agB.count):=
case
agB.state=receive3beta & agB.count<2|agB.state=receive3alpha & agB.count<2: agB.count+1;
agB.state=send4alpha |agB.state=send4beta:agB.count;
agB.count=1 & agA.state=receive4alpha & !Iactive:0;
agB.count=2 & agA.state=send3alpha|agB.count=1 & agA.state=send3alpha: 0;
1:agB.count;
esac;

--Transitions for Agent I's counter

next(agI.count):=
case
agI.state=receive3beta & agI.count<2 & Iactive:agI.count+1;
agI.state=send3beta & Iactive:agI.count;
agI.state=receive4beta & agI.count<2 & Iactive: agI.count+1;
agI.count=2: 0;
1:agI.count;
esac;

--Transitions for variable indicating agent A's authentication
```

```
next(agA.authenticated):=
case
agA.state=receive4alpha :TRUE;
1:agA.authenticated;
esac;

--Transitions for variable indicating agent B's authentication

next(agB.authenticated):=
case
agB.state=send4alpha |agB.state=send4beta :TRUE;
1:agB.authenticated;
esac;

--Transitions for variable indicating agent B's authentication which
--indicates that it has received the fourth message

next(agI.authenticated):=
case
agI.state=receive4beta :TRUE;
1:agI.authenticated;
esac;

--Agent S always is authenticated so transitions to the false state do not occur

next(agS.authenticated):=
case
1:agS.authenticated;
esac;

--Specifications which detect the presence of replay attack

--Agent B should not receive more messages than what agent A has sent it
--SPEC AG!(agA.count < agB.count);

--Agent I should never receive the fourth message
SPEC AG!(agI.state=receive4beta);

--Module for each agent's type which includes the agent's state variable,
--its counter and its authentication variable

MODULE agtype
```

```
VAR
state: {wait, send1, receive1, send2,receive2,
send3alpha, send3beta, receive3alpha, receive3beta,
send4alpha,send4beta, receive4alpha, receive4beta };
count:{0,1,2};
authenticated:boolean;
```

A.2 Kerberos Protocol with Freshness Concept

```
MODULE main

VAR

--Creating agents which are to type agtype
agA : agtype;
agB : agtype;
agS : agtype;
agI : agtype;
Iactive: boolean;
Fresh:0..20;
Time:0..20;

--Assigning initial to values to all variables

ASSIGN
init(agA.state):=wait;
init(agB.state):=wait;
init(agS.state):=wait;
init(agI.state):=wait;
init(agA.count):=0;
init(agB.count):=0;
init(agS.count):=0;
init(agI.count):=0;
init(agA.authenticated):=FALSE;
init(agB.authenticated):=FALSE;
init(agI.authenticated):=FALSE;
init(agS.authenticated):=TRUE;
init(Fresh):=0;
init(Time):=0;
```

```
--Transitions for the variable indicating presence or absence of intruder

next(Iactive):=
case
!Iactive:{0,1};
Iactive & agI.state=receive4beta:{0,1};
1:Iactive;
esac;

--Transitions for agent A's state

next(agA.state):=
case
agA.state=wait: send1;
agS.state=send2 & agA.state=send1: receive2;
agA.state=receive2: send3alpha;
agB.state=send4alpha & agA.state=send3alpha: receive4alpha;
agA.state=receive4alpha: wait;
1:agA.state;
esac;

--Transitions for agent B's state

next(agB.state):=
case
agA.state=send3alpha & agB.state=wait & Fresh=0: receive3alpha;
agI.state=send3beta & agB.state=wait & Iactive & Fresh=0: receive3beta;
agI.state=send3beta & agB.state=send4alpha & Iactive & Fresh=0: receive3beta;
agB.state=receive3alpha:send4alpha;
agB.state=receive3beta:send4beta;
agB.state=send4alpha:wait;
1:agB.state;
esac;

--Transitions for Server S's state

next(agS.state):=
case
agA.state=send1 & agS.state=wait: receive1;
agS.state=receive1:send2;
agS.state=send2:wait;
1:agS.state;
esac;
```

```
--Transitions for the Intruder's state

next(agI.state):=
case
agI.state=wait & agA.state=send3alpha & agB.state=wait & Iactive: receive3beta;
agI.state=receive3beta & Iactive: send3beta;
agI.state=send3beta & agB.state=send4beta & Iactive  : receive4beta;
agI.state=receive4beta & Iactive: wait;
agI.state=send3beta & Time>2: wait;
1:agI.state;
esac;

--Transitions for Agent A's counter

next(agA.count):=
case
agA.state=send1|agA.state=receive2: agA.count;
agA.state=send3alpha & agA.count<1:agA.count+1;
agA.count=1 & agA.state=receive2: 0;
1:agA.count;
esac;

--Transitions for Agent B's counter

next(agB.count):=
case
agB.state=receive3beta & agB.count<2|agB.state=receive3alpha & agB.count<2: agB.count+1;
agB.state=send4alpha|agB.state=send4beta:agB.count;
agB.count=1 & agA.state=receive4alpha & !Iactive:0;
agB.count=2 & agA.state=send3alpha|agB.count=1 & agA.state=send3alpha: 0;
1:agB.count;
esac;

--Transitions for Agent I's counter

next(agI.count):=
case
agI.state=receive3beta & agI.count<2 & Iactive:agI.count+1;
agI.state=send3beta & Iactive:agI.count;
agI.state=receive4beta & agI.count<2 & Iactive: agI.count+1;
agI.count=2: 0;
1:agI.count;
esac;
```

```
--Transitions for variable indicating agent A's authentication

next(agA.authenticated):=
case
agA.state=receive4alpha :TRUE;
1:agA.authenticated;
esac;

--Transitions for variable indicating agent B's authentication

next(agB.authenticated):=
case
agB.state=send4alpha|agB.state=send4beta :TRUE;
1:agB.authenticated;
esac;

--Transitions for variable indicating agent B's authentication which
--indicates that it has received the fourth message

next(agI.authenticated):=
case
agI.state=receive4beta :TRUE;
1:agI.authenticated;
esac;

--Agent S always is authenticated so transitions to the false state do not occur

next(agS.authenticated):=
case
1:agS.authenticated;
esac;

--Transitions for the freshness variable

next(Fresh):=
case
agA.state=send3alpha & agB.state=wait:0;
Fresh<20:Fresh+1;
1:0;
esac;

--Transitions for the Intruder's timer variable

next(Time):=
```

```
case
agI.state=receive3beta:0;
Time<20:Time+1;
1:0;
esac;

--Specifications which detect the presence of replay attack

--Agent B should not receive more messages than what agent A has sent it
SPEC AG!(agA.count < agB.count);

--Agent I should never receive the fourth message
SPEC AG!(agI.state=receive4beta);

--Module for each agent's type which includes the agent's state variable,
--its counter and its authentication variable

MODULE agtype

VAR
state:{wait, send1, receive1, send2,receive2,
send3alpha, send3beta, receive3alpha, receive3beta,
send4alpha, send4beta, receive4alpha, receive4beta};
count:{0,1,2};
authenticated:boolean;
```

Appendix B

Trace Files

B.1 Replay Attack

```
-- specification AG !(agA.count < agB.count)  is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
-> State: 1.1 <-
  agA.state = wait
  agA.count = 0
  agA.authenticated = 0
  agB.state = wait
  agB.count = 0
  agB.authenticated = 0
  agS.state = wait
  agS.count = 0
  agS.authenticated = 1
  agI.state = wait
  agI.count = 0
  agI.authenticated = 0
  Iactive = 0
-> Input: 1.2 <-
-> State: 1.2 <-
  agA.state = send1
  agS.count = 2
-> Input: 1.3 <-
-> State: 1.3 <-
  agS.state = receive1
-> Input: 1.4 <-
-> State: 1.4 <-
```

```
  agS.state = send2
-> Input: 1.5 <-
-> State: 1.5 <-
  agA.state = receive2
  agS.state = wait
-> Input: 1.6 <-
-> State: 1.6 <-
  agA.state = send3alpha
  Iactive = 1
-> Input: 1.7 <-
-> State: 1.7 <-
  agA.count = 1
  agB.state = receive3alpha
  agI.state = receive3beta
-> Input: 1.8 <-
-> State: 1.8 <-
  agB.state = send4alpha
  agB.count = 1
  agI.state = send3beta
  agI.count = 1
-> Input: 1.9 <-
-> State: 1.9 <-
  agA.state = receive4alpha
  agB.state = receive3beta
  agB.authenticated = 1
-> Input: 1.10 <-
-> State: 1.10 <-
  agA.state = wait
  agA.authenticated = 1
  agB.state = send4beta
  agB.count = 2
-- specification AG !(agI.state = receive4beta)  is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
-> State: 2.1 <-
  agA.state = wait
  agA.count = 0
  agA.authenticated = 0
  agB.state = wait
  agB.count = 0
  agB.authenticated = 0
  agS.state = wait
  agS.count = 0
```

```
  agS.authenticated = 1
  agI.state = wait
  agI.count = 0
  agI.authenticated = 0
  Iactive = 0
-> Input: 2.2 <-
-> State: 2.2 <-
  agA.state = send1
  agS.count = 2
-> Input: 2.3 <-
-> State: 2.3 <-
  agS.state = receive1
-> Input: 2.4 <-
-> State: 2.4 <-
  agS.state = send2
-> Input: 2.5 <-
-> State: 2.5 <-
  agA.state = receive2
  agS.state = wait
-> Input: 2.6 <-
-> State: 2.6 <-
  agA.state = send3alpha
  Iactive = 1
-> Input: 2.7 <-
-> State: 2.7 <-
  agA.count = 1
  agB.state = receive3alpha
  agI.state = receive3beta
-> Input: 2.8 <-
-> State: 2.8 <-
  agB.state = send4alpha
  agB.count = 1
  agI.state = send3beta
  agI.count = 1
-> Input: 2.9 <-
-> State: 2.9 <-
  agA.state = receive4alpha
  agB.state = receive3beta
  agB.authenticated = 1
-> Input: 2.10 <-
-> State: 2.10 <-
  agA.state = wait
  agA.authenticated = 1
  agB.state = send4beta
```

```
  agB.count = 2
-> Input: 2.11 <-
-> State: 2.11 <-
  agA.state = send1
  agI.state = receive4beta
```

B.2 Replay Attack overcome using Freshness Concept

```
-- specification AG !(agA.count < agB.count)  is true
-- specification AG !(agI.state = receive4beta)  is true
```

References

[1] H. Debar, M. Dacier, and A. Wespi, "A revised taxonomy for intrusion-detection systems," *Annales Des Télécommunications*, vol. 55, no. 7, pp. 361–378, Jul 2000. [Online]. Available: https://doi.org/10.1007/BF02994844

[2] P. V. S. Alpao, J. R. I. Pedrasa, and R. Atienza, "Multilayer perceptron with binary weights and activations for intrusion detection of cyber-physical systems," in *TENCON 2017 - 2017 IEEE Region 10 Conference*, Nov 2017, pp. 2825–2829.

Table B.1: Project Detail

Student Details

Student Name	Your Name		
Registration Number	070911001	Section/Roll No.	A/01
Email Address	baravkar.nikhil05@gmail.com	Phone No.(M)	9891000000
Student Name	Your Name		
Registration Number	070911001	Section/Roll No.	A/01
Email Address	yourname@yahoo.com	Phone No.(M)	9891000000

Project Details

Project Title	Title of your project		
Project Duration	4-6 Months	Date of Reporting	14-01-2011

Organization Details

Organization Name	Name of your organization
Full Postal Address	Whitefield, B'lore
Website Address	www.abc.com

Supervisor Details

Supervisor Full Name	Name		
Designation	Project Leader or Manager		
Full Contact Address with PIN Code	#1,Whitefield, B'lore		
Email Address	xyz@abc.in	Phone No.(M)	9767541234

Internal Guide Details

Faculty Name	Name
Full Contact Address with PIN Code	Department of Information and Communication Technology, Manipal Institute of Technology, Manipal-576104
Email Address	abc@manipal.edu