

Air Force Institute of Technology

AFIT Scholar

Theses and Dissertations

Student Graduate Works

3-26-2015

Evaluating Machine Learning Classifiers for Hybrid Network Intrusion Detection Systems

Michael D. Rich

Follow this and additional works at: <https://scholar.afit.edu/etd>



Part of the [Information Security Commons](#)

Recommended Citation

Rich, Michael D., "Evaluating Machine Learning Classifiers for Hybrid Network Intrusion Detection Systems" (2015). *Theses and Dissertations*. 52.
<https://scholar.afit.edu/etd/52>

This Thesis is brought to you for free and open access by the Student Graduate Works at AFIT Scholar. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of AFIT Scholar. For more information, please contact AFIT.ENWL.Repository@us.af.mil.



**EVALUATING MACHINE LEARNING
CLASSIFIERS FOR HYBRID NETWORK
INTRUSION DETECTION SYSTEMS**

THESIS

Michael D. Rich, MSgt, USAF
AFIT-ENG-MS-15-M-046

**DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY**

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

DISTRIBUTION STATEMENT A
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

The views expressed in this document are those of the author and do not reflect the official policy or position of the United States Air Force, the United States Department of Defense or the United States Government. This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

AFIT-ENG-MS-15-M-046

EVALUATING MACHINE LEARNING CLASSIFIERS FOR HYBRID
NETWORK INTRUSION DETECTION SYSTEMS

THESIS

Presented to the Faculty
Department of Electrical and Computer Engineering
Graduate School of Engineering and Management
Air Force Institute of Technology
Air University
Air Education and Training Command
in Partial Fulfillment of the Requirements for the
Degree of Master of Science in Cyber Operations

Michael D. Rich, B.S.

MSgt, USAF

March 2015

DISTRIBUTION STATEMENT A
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

AFIT-ENG-MS-15-M-046

EVALUATING MACHINE LEARNING CLASSIFIERS FOR HYBRID
NETWORK INTRUSION DETECTION SYSTEMS

THESIS

Michael D. Rich, B.S.
MSgt, USAF

Committee Membership:

Robert F. Mills, PhD (Chairman)

Steven K. Rogers, PhD (Member)

Maj Thomas E. Dube, PhD (Member)

Abstract

Existing classifier evaluation methods do not fully capture the intended use of classifiers in hybrid intrusion detection systems (IDS), systems that employ machine learning alongside a signature-based IDS. This research challenges traditional classifier evaluation methods in favor of a value-focused evaluation method that incorporates evaluator-specific weights for classifier and prediction threshold selection. By allowing the evaluator to weight known and unknown threat detection by alert classification, classifier selection is optimized to evaluator values for this application. The proposed evaluation methods are applied to a Cyber Defense Exercise (CDX) dataset. Network data is processed to produce connection-level features, then labeled by correlating packet-level alerts from a signature-based IDS. Seven machine learning algorithms are evaluated using traditional methods and the value-focused method. Comparing results from the two evaluation methods, fallacies are revealed with 2 of the 5 notional weighting schemes that would lead to suboptimal classifier and prediction threshold selection when using traditional methods that do not consider evaluator values.

Acknowledgements

First and foremost, I must thank my family. My wife and daughters give me the motivation to persevere through any challenge. I do this for them, because of their relentless support, love, and encouragement.

I give a sincere thank you to my research advisor, Dr. Robert Mills, for his guidance and flexibility allowing me to research a topic of interest. I would like to thank my sponsor and committee member, Dr. Steve Rogers, for the countless discussions that extend well beyond the research presented in this thesis. I would also like to thank my committee member, Maj Thomas Dube, for his knowledge and support with the technical aspects and results analysis of my research. My time as an AFIT student has been an incredible experience because of the great mentorship received from my thesis committee, other faculty members, and students.

Lastly, I would like to thank my supervisors and leadership from previous assignments for guiding my career and pushing me beyond my own self-imposed limitations.

Michael D. Rich

Table of Contents

	Page
Abstract	iv
Acknowledgements	v
List of Figures	ix
List of Tables	xi
List of Acronyms	xiii
I. Introduction	1
1.1 Research Goals and Hypothesis	3
1.2 Research Contributions	4
1.3 Preview	4
II. Related Research	5
2.1 Chapter Overview	5
2.2 Intrusion Detection Systems	5
2.2.1 Snort	6
2.2.2 Bro	9
2.3 Intrusion Detection Methodologies	12
2.3.1 Misuse Detection	13
2.3.2 Anomaly-Based Detection	14
2.3.3 Stateful Protocol Analysis	14
2.3.4 Hybrid Detection	15
2.4 Machine Learning and Intrusion Detection	18
2.4.1 General Network Traffic Classification	19
2.4.2 Network Threat Detection Classification	20
2.5 Classifiers	25
2.5.1 Bayesian Network	25
2.5.2 Instance-Based Learner (IB k)	26
2.5.3 Sequential Minimal Optimization (SMO)	26
2.5.4 Decision Trees	27
2.5.5 Artificial Neural Networks (ANNs)	27
2.5.6 Boosting Algorithms	28
2.6 Summary	28

	Page
III. Methodology	30
3.1 Chapter Overview	30
3.2 Proposed System	30
3.3 Dataset Description	31
3.4 Network Sensor	32
3.5 Data Preprocessing	36
3.5.1 Processing Network Packet Capture Data	36
3.5.2 Connection-Level Threat Labeling	39
3.5.3 Data Cleanup	43
3.5.4 Dataset Subsets	44
3.5.5 Additional Features	45
3.5.6 Data Balancing	45
3.5.7 Final Dataset Descriptions	46
3.6 WEKA Configuration	46
3.7 Experiments	47
3.7.1 Classifier Performance	48
3.7.2 Unknown Threat Detection	49
3.8 Performance Metrics	51
3.8.1 Confusion Matrix	52
3.8.2 Marginal Rates	52
3.8.3 Overall Rates	54
3.8.4 Precision-Recall Curves	56
3.8.5 Unknown Threat Detection Rate	58
3.9 Value Focused Thinking (VFT) Evaluation Approach	58
3.9.1 Applying VFT to Intrusion Detection	59
3.9.2 Known Threat Detection	62
3.9.3 Unknown Threat Detection	63
3.9.4 Precision	64
3.9.5 VFT Score Calculation	65
3.10 Summary	66
IV. Results and Analysis	67
4.1 Chapter Overview	67
4.2 Classification Results by Dataset	67
4.2.1 All Network Data	68
4.2.2 TCP Data	70
4.2.3 HTTP Data	73
4.2.4 HTTP Data with Additional Features	75
4.3 Value Focused Thinking Evaluation Results	78
4.3.1 PR Curves by Alert Class	79
4.3.2 Scenario 1 - Balanced	81
4.3.3 Scenario 2 - Alert Class Priority	83

	Page
4.3.4 Scenario 3 - Detect Unknown Threats	86
4.3.5 Scenario 4 - Confirm Known Threats	88
4.3.6 Scenario 5 - High Precision	90
4.3.7 VFT Results Summary	92
4.4 Summary	93
V. Conclusion	96
5.1 Chapter Overview	96
5.2 Conclusions of Research	96
5.3 Research Contributions	97
5.4 Recommendations for Future Research	98
Appendix A. Default Snort Alert Classifications	99
Appendix B. Network Sensor Configuration	101
Appendix C. WEKA Commands and Options	105
Appendix D. Confusion Matrices	109
Bibliography	111

List of Figures

Figure	Page
2.1. Snort Architecture and Data Flow	7
2.2. Example Snort Rule	8
2.3. Structure of Bro System [34]	10
2.4. Anomaly-Misuse Detection System Workflow [16, 44]	16
2.5. Misuse-Anomaly Detection System Workflow [16, 49]	17
2.6. Parallel Detection System Workflow [16]	18
3.1. Proposed System Architecture	31
3.2. CDX Network Architecture [32]	33
3.3. Example tcpreplay Output	36
3.4. CDX Data Packet-Level Alert Classification Metrics	37
3.5. CDX Data Packet-Level Alert Priority Metrics	38
3.6. CDX Data Connections by Service	39
3.7. Alert Dictionary Algorithm	40
3.8. Connection-Alert Correlation and Labeling Algorithm	41
3.9. CDX Data Connection-Level Threats by Service	43
3.10. Unknown Threat Dataset Creation Algorithm	50
3.11. Comparing Algorithms in ROC vs PR Space [14]	57
3.12. Calculating Figures of Merit (adapted from [27])	60
3.13. Calculating FOM with Classifier Thresholds	61
3.14. VFT Intrusion Detection Hierarchy	62
4.1. Classifier PR Curves - All Network Data	70
4.2. Classifier PR Curves - TCP Data	72

Figure	Page
4.3. Classifier PR Curves - HTTP Data	75
4.4. Classifier PR Curves - HTTP Data with Additional Features	77
4.5. PR Curves by Alert Class - Classifier Performance	79
4.6. PR Curves by Alert Class - Unknown Threat Detection	80
4.7. VFT Optimal Thresholds - Scenario 1	83
4.8. VFT Optimal Thresholds - Scenario 2	85
4.9. VFT Optimal Thresholds - Scenario 3	88
4.10. VFT Optimal Thresholds - Scenario 4	90
4.11. VFT Optimal Thresholds - Scenario 5	92
B.1. Checking Status of Security Onion Services	102
B.2. Disabling Daily Rule Update Script	103

List of Tables

Table	Page
2.1. Bro conn.log Fields [2]	11
2.2. Bro conn_state Meanings [2]	12
2.3. Bro history Meanings [2]	13
2.4. Summary of Related Work	25
3.1. CDX Data Set Description by Year	32
3.2. Unified2 Log Entry Fields [6]	34
3.3. Additional Connection Log Features	35
3.4. Final Dataset Descriptions	46
3.5. Two-Class Confusion Matrix Example	53
4.1. Mean Overall Rates with 95% CI - All Network Data	68
4.2. Mean Marginal Rates with 95% CI - All Network Data	69
4.3. Mean Overall Rates with 95% CI - TCP Data	70
4.4. Mean Marginal Rates with 95% CI - TCP Data	71
4.5. Performance Comparison of TCP-Only Connections	72
4.6. Mean Overall Rates with 95% CI - HTTP Data	73
4.7. Mean Marginal Rates with 95% CI - HTTP Data	74
4.8. Mean Overall Rates with 95% CI - HTTP Data with Additional Features	76
4.9. Mean Marginal Rates with 95% CI - HTTP Data with Additional Features	76
4.10. Performance Comparison of HTTP-Only Connections	78
4.11. VFT Weighting Scheme - Scenario 1	81
4.12. VFT Evaluation Results - Scenario 1	82

Table	Page
4.13.	VFT Weighting Scheme - Scenario 2 84
4.14.	VFT Evaluation Results - Scenario 2 84
4.15.	VFT Weighting Scheme - Scenario 3 86
4.16.	VFT Evaluation Results - Scenario 3 87
4.17.	VFT Weighting Scheme - Scenario 4 89
4.18.	VFT Evaluation Results - Scenario 4 89
4.19.	VFT Weighting Scheme - Scenario 5 91
4.20.	VFT Evaluation Results - Scenario 5 91
4.21.	VFT Weighting Schemes and Results Summary 93
A.1	Default Snort Alert Classifications [6] 99
B.1.	Security Onion Virtual Machine Hardware Settings 101
B.2.	Security Onion Applications 102
D.1.	Confusion Matrix - All Network Data 109
D.2.	Confusion Matrix - TCP Data 109
D.3.	Confusion Matrix - HTTP Data 109
D.4.	Confusion Matrix - HTTP Data with Additional Features 110
D.5.	Confusion Matrix - FTP Data 110
D.6.	Confusion Matrix - FTP Data with Additional Features 110

List of Acronyms

Acronym	Page
IDS	Intrusion Detection Systems 1
TCP	Transmission Control Protocol 3
IP	Internet Protocol 3
VFT	Value-Focused Thinking 4
PR	Precision-Recall 4
NIST	National Institute of Standards and Technology 5
NIDS	Network-Based Intrusion Detection System 5
HIDS	Host-Based Intrusion Detection System 5
HTTP	Hypertext Transfer Protocol 7
SMTP	Simple Mail Transfer Protocol 7
FTP	File Transfer Protocol 7
POP3	Post Office Protocol v3 7
ET	Emerging-Threats 7
VRT	Vulnerability Research Team 7
SQL	Structure Query Language 8
UDP	User Datagram Protocol 9
ICMP	Internet Control Message Protocol 9
QoS	Quality of Service 19
DARPA	Defense Advanced Research Project Agency 21
AFRL	Air Force Research Laboratory 21
AdaBoost	Adaptive Boosting 23
LEM	Laplacian Eigenmap 24

Acronym		Page
RLS	Regularized Least Squares	24
SVM	Support Vector Machine	24
WEKA	Waikato Environment for Knowledge Analysis	25
SMO	Sequential Minimal Optimization	26
ANNs	Artificial Neural Networks	27
MLP	Multilayer Perceptron	28
CDX	Cyber Defense Exercise	31
NSA	National Security Agency	31
NPS	Naval Postgraduate School	32
AFIT	Air Force Institute of Technology	32
ARFF	Attribute-Relation File Format	43
RFC	Request for Comments	45
API	Application Program Interface	47
TP	True Positive	52
TN	True Negative	52
FP	False Positive	52
FN	False Negative	52
TPR	True Positive Rate	53
TNR	True Negative Rate	53
FPR	False Positive Rate	53
FNR	False Negative Rate	54
CI	Confidence Interval	54
MCC	Matthews Correlation Coefficient	56
ROC	Receiver Operating Characteristic	56

Acronym		Page
FOM	Figures of Merit	59
UTD	Unknown Threat Detection	69

EVALUATING MACHINE LEARNING CLASSIFIERS FOR HYBRID NETWORK INTRUSION DETECTION SYSTEMS

I. Introduction

Intrusion detection is often the primary triggering mechanism for network security analysts to respond to malicious activities on a computer network. The volume and velocity of communications on today's networks require the use of automated Intrusion Detection Systems (IDS) as human analysts cannot review all transmitted information manually. IDS alerts are typically the starting point for an analyst's review of an intrusion. After the initial alert, an analyst performs investigations into raw network data, system log sources, and other sources to confirm or deny the alert. An IDS often produces false alarms and misses threats. False alarms cause undue workload on analysts and missed threats leave the organization in a state of false security as intruders unknowingly infiltrate their networks.

The current cyber environment is littered with intrusions and data breaches. The 2013 Data Breach Investigations Report by Verizon analyzed 47,000 security incidents and 621 confirmed data breaches and indicated that 66% of the 621 data breaches "took months or more to discover" [45]. This high percentage of data breaches with delayed detection times reveals one of many problems with the operationalization of IDSes. Other challenges in the intrusion detection problem include false alarms, missed attacks, and the detection of unknown attacks.

Misuse detection, the use of signatures for known threats, is the primary method used in today's IDSes. As vulnerabilities are discovered, signatures are created and used for detection. This method is reactive and is not capable of detecting unknown

threats, those threats which the detection system lacks a corresponding detection signature. Research on anomaly-based intrusion detection systems has been ongoing since the 1980's but has failed to establish its place in the operational environment that is dominated by signature-based systems. This is primarily attributed to high false positive rates from these anomaly-based systems. Being that signature-based systems cannot detect unknown threats, it is necessary that user confidence for anomaly-based systems improves and operational use of these systems increases. Hybrid solutions, those that incorporate both a misuse and anomaly detector, have been shown to provide many benefits such as reducing false positives and negatives, supporting verification of true and false positives, and detecting unknown threats.

Deploying an anomaly-based IDS in a hybrid configuration alongside an existing signature-based IDS is the next logical advancement in IDS technologies, allowing an organization to enhance an existing signature-based network sensor with the capability of detecting unknown threats. This is a classification problem that is commonly addressed using machine learning classifiers. Much research has been done over the past 15 years with machine learning in the intrusion detection domain. Yet, there is no clear method of selecting an optimal classifier, or set of classifiers, for use in a hybrid IDS architecture. While the “no free lunch” theorem [47] explains the implications of optimization, this research aims to fill the gap of selecting an optimal classifier in terms of evaluator-specified values by providing a methodology for evaluating classifiers for specific use in hybrid IDS systems. The intent is to provide an evaluation method that will improve user confidence in the use of machine learning classifiers alongside existing signature-based IDSes.

1.1 Research Goals and Hypothesis

The overall goal of this research is to develop a methodology of evaluating a threat classifier system to use in a hybrid IDS architecture. Emphasis is placed on detecting unknown threats while minimizing false positives and negatives as these are common goals among all organizations. Threat classification is accomplished at the connection-level with experiments at multiple layers of the Transmission Control Protocol (TCP)/Internet Protocol (IP) stack to determine the appropriate layer to generalize network traffic. The proposed methodology should allow an organization to develop and test multiple machine learning classifiers in a sensor-specific manner, tailoring the selection to the organization's own data and network services. The ability to tailor classifier selection based on other organization-specific goals, resources, and security requirements should also be included. As these goals and requirements change over time, the evaluator should be able to quickly reevaluate classifiers. An evaluator should be able to weight parameters for detecting known and unknown threats, false positives, and detection rates by type of attack. Specific research goals are:

1. Provide a machine learning classifier evaluation methodology which incorporates a weighted scheme allowing evaluator-specific requirements to be considered in the evaluation process.
2. Demonstrate an engineering advantage by applying the methodology to evaluate multiple classifiers used in intrusion detection experiments using notional weighting schemes.
3. Conduct experiments at various layers in the TCP/IP stack to determine if the layer in which network traffic is generalized has any statistical significance in results.

The hypothesis is that there will be an engineering advantage to incorporating an evaluator-specific weighting scheme into the classifier evaluation process. While not measured, user confidence for selected machine learning classifiers should increase by selecting classifiers developed that align with user values.

1.2 Research Contributions

This research contributes to the field of intrusion detection, with an emphasis in the application of machine learning. This research presents a comprehensive approach, similar to cross-validation, to evaluate a classifier’s ability to detect unknown threats by allowing every malicious connection in the dataset the opportunity to be tested as an unknown threat. This research also applies the Value-Focused Thinking (VFT) decision-making approach, from the field of operations research, to evaluate classifiers using a value hierarchy model constructed towards using the classifiers in a hybrid IDS architecture. Lastly, this research presents a unique approach to parsing a traditional Precision-Recall (PR) curve into multiple curves representing different attack classifications allowing the classification performance for each classifier to be compared by attack class across the entire range of prediction thresholds.

1.3 Preview

This chapter introduced problems within the intrusion detection domain, focusing on the operationalization of machine learning in hybrid IDS architectures. The research goals, hypotheses, and contributions were also presented. Chapter II covers related work to include a background of IDS technologies and the application of machine learning to intrusion detection. Chapter III explains the methodologies used for the experiments in this research. Experiment results are in Chapter IV. Conclusions are provided in Chapter V along with recommendations for future research.

II. Related Research

2.1 Chapter Overview

This chapter provides a background of IDS technologies and how machine learning is being applied to the problem. The first section covers a brief history of IDSes and two popular open-source IDS applications. The following section examines intrusion detection methodologies, thoroughly explaining the difference between misuse and anomaly detection and how they can be applied in a hybrid configuration. Next, a literature review of recent research on the application of machine learning to network traffic and threat detection classification is presented. Finally, a brief technical background of the machine learning classifiers used in this thesis is provided.

2.2 Intrusion Detection Systems

An intrusion is “the act of wrongfully entering upon, seizing, or taking possession of the property of another” [4]. Adapting this definition to the cyber domain, an intrusion can be considered any attempt to compromise a system’s confidentiality, integrity, or availability. The early works of detecting intrusions in the cyber domain can be traced back to the 1980’s with the seminal research conducted by Anderson [11] and Denning [15]. The detection methodology shifted from primarily manual inspection of audit logs to automated systems that analyze large amounts of data in real-time.

The National Institute of Standards and Technology (NIST) categorizes IDSes by the method used to collect and analyze audit data [40]. A Network-Based Intrusion Detection System (NIDS) analyzes network traffic for suspicious behavior. A Host-Based Intrusion Detection System (HIDS) monitors activity on a specific host. A wireless IDS analyzes wireless network traffic for suspicious behavior. Finally, a

network behavior analysis system analyzes statistical features of network traffic to identify suspicious behavior.

Many challenges exist in intrusion detection. Intrusion detection is ultimately a classification problem. Audit data is collected, analyzed, and classified as malicious or normal. As with any other classification problem, false positives and negatives are a challenge. Other challenges specific to network intrusion detection are constantly evolving environments, detection of novel attacks, and encryption. Research in the IDS domain is limited by lack of ground truth data and standardized datasets to evaluate IDS technologies.

This research is restricted to network-based IDS. The following sections provide in depth detail of two popular open-source NIDS, Snort and Bro, both of which are used in the research experiments of this thesis.

2.2.1 Snort.

Originally released in 1999, Snort built upon tcpdump, a packet sniffing tool, adding the ability for packet payload inspection using easy to write rules [38]. Snort has since become the leading open source signature-based IDS. The architecture showing the data flow through Snort is shown in Figure 2.1. Each process within the architecture is now briefly described:

- *Libpcap Library*: An open source, UNIX-based, library that provides an application programming interface (API) to capture network packet data. Designed to be used with C or C++, but wrappers are available to access the API with languages like Perl, Python, Java, C#, or Ruby [20]. Winpcap is the Windows equivalent.
- *Packet Decoder*: Receives the Ethernet frame from libpcap and analyzes packet headers for IP and transport layers. Decoder rules can be configured to per-

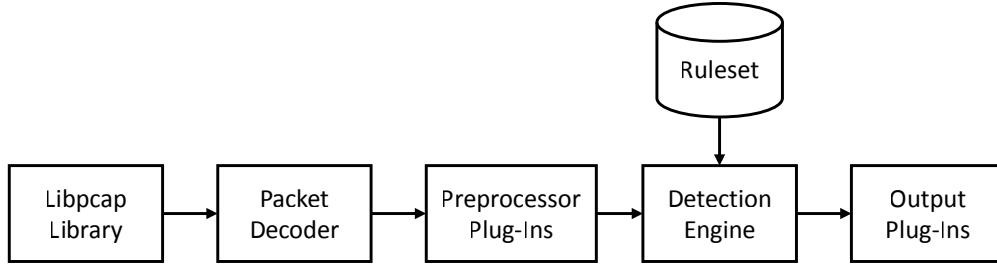


Figure 2.1. Snort Architecture and Data Flow

form actions at this step if desired, such as dropping a packet prior to it being forwarded to the preprocessor.

- *Preprocessor Plug-Ins*: Modular plug-ins that run after each packet is decoded, but prior to the detection engine [6]. Plug-ins are used for most application-level services (e.g., Hypertext Transfer Protocol (HTTP), Simple Mail Transfer Protocol (SMTP), File Transfer Protocol (FTP), Post Office Protocol v3 (POP3)) in order to process application-layer header information prior to the detection engine. This allows rules to be written using protocol-specific fields. Other plug-ins are used for specific functionality, such as inspecting fragmented packets and detection of port scan activities.
- *Detection Engine*: The process used to compare each packet with the rules stored in the ruleset. A simplified view of this process is that every packet is compared to every rule in the ruleset. In fact, there is a complex data structure consisting of chains of rules and options that are used to compare rules to packet contents. Many improvements to this process have been researched to increase the speed of detections and minimize packet drops on high bandwidth interfaces. Specific details of this process is beyond the scope of this thesis.
- *Ruleset*: Repository of rules used to perform packet inspection. Emerging-Threats (ET) [3], an open source ruleset library maintained by the Snort community, and the Talos (formerly the Vulnerability Research Team (VRT)) [7]

ruleset libraries are the two leading rulesets in use. Both libraries offer paid subscription and free versions of their rulesets, the difference being less delay in receiving newly developed rules for paid users. Users also have the ability to create site specific rules in addition to these public rulesets. Rules are stored in flat-text files.

- *Output Plug-Ins*: Modular plug-ins that enable alerts to be output in various formats. Alerts can be output as a simple text file or formatted for an Structure Query Language (SQL) database, syslog, or other desired format. The modularity of the output plug-ins allows for customization to any user's network security monitoring architecture.

Snort operates by inspecting individual packets using rules that match patterns of bytes. An example Snort rule is provided in Figure 2.2. This rule sends an alert when a TCP packet is detected from `$EXTERNAL_NET` any port, to `$HTTP_SERVERS` `$HTTP_PORTS` meeting the conditions specified within the parenthesis. `$EXTERNAL_NET`, `$HTTP_SERVERS`, and `$HTTP_PORTS` are dynamic variables that are set in the Snort configuration file. The alert message "ET WEB_SERVER cmd.exe In URI - Possible Command Execution Attempt" is sent when a TCP connection is established and the content of `/cmd.exe` is detected in the `http_uri` field of the HTTP header. The reference document is noted along with the class type, signature identification number, and signature revision number. This signature ultimately alerts when `/cmd.exe` is found in a specific type of packet under specific conditions that have been deemed to

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"ET \
WEB_SERVER cmd.exe In URI - Possible Command Execution Attempt"; \
flow:to_server,established; content:"/cmd.exe"; fast_pattern:only; \
nocase; http_uri; reference:url,doc.emergingthreats.net/2009361; \
classtype:attempted-recon; sid:2009361; rev:6;)
```

Figure 2.2. Example Snort Rule

represent a command execution attempt. This is a rather simple rule using a small set of the options available in Snort. A full description of options with more rule examples can be found in the Snort Users Manual [6].

Alert classification types are commonly used for management of rules and to aid in alert analysis. By default, the priority of each rule is set to the corresponding alert classification. Priorities can be modified per rule if desired. Example alert classifications are network scans, attempted denial-of-service, and shellcode detection. The complete list of default alert classifications and priorities are listed in Appendix A. The use of alert classifications will be leveraged in the experiments of this thesis.

2.2.2 Bro.

Bro is an open-source network security monitor originally developed by Paxson [34] and currently maintained as a joint team of researchers and developers at the International Computer Science Institute in Berkeley, CA and the National Center for Supercomputing Applications in Urbana-Champaign, IL [1]. Bro incorporates a separation of mechanism and policy that does not inherit a “good” vs “bad” ideology like a typical IDS, but instead leaves the user to define a local policy designed around network-based events. Instead of loading rules that detect malicious activity, all network data is monitored and logged, then scripts are used to take action when specific criteria are met.

Bro also differs from a typical IDS in that it is connection-based rather than packet-based, meaning that events are related to connections where a typical IDS such as Snort relates alerts to packets. A connection can contain one or many packets. Bro’s definition of a connection includes TCP connections as well as User Datagram Protocol (UDP) and Internet Control Message Protocol (ICMP) flows. TCP connections in Bro are created when the first packet of an unknown connection, a

connection not actively being monitored, is seen and closed when the TCP connection is terminated or a configured timeout interval has been reached. UDP and ICMP connections are created when the first packet of an unknown connection is seen and closed when a configured timeout interval has been reached. UDP and ICMP packets are related to actively monitored connections if they contain matching source and destination IP addresses and ports and arrive before the timeout interval has elapsed. Every connection receives a unique identifier.

The architecture of Bro is presented in Figure 2.3. Like Snort, network data is first filtered using libpcap. The filtered network data is processed by the Event Engine which abstracts events from the network data. Example events are `connection_established`, `http_request`, `http_reply`, and `connection_finished`. Protocol analyzers are included for all common networking protocols which define each event and the information contained within each event. The event stream is then processed by the Policy Script Interpreter which contains a set of scripts that take action based on the events received. Actions could include logging information, generating alerts,

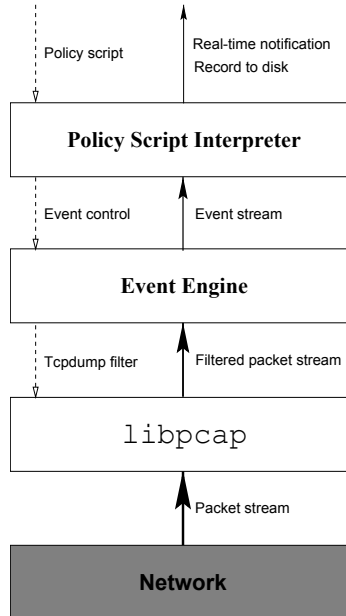


Figure 2.3. Structure of Bro System [34]

or executing additional scripts or programs. Default policy scripts are included with Bro to provide an initial logging and malicious detection capabilities. The default logs produced by Bro are the conn.log, http.log, dns.log, dhcp.log, ftp.log, smtp.log, and others. Each log contains specific features related to connections of a specific type. Users are also able to develop their own local policy scripts using the Bro programming language to produce logs and alerts as desired.

The conn.log is largest of the default logs as it contains the details on every con-

Table 2.1. Bro conn.log Fields [2]

Feature	Data Type	Description
ts	time	Time of first packet
uid	string	A unique identifier of the connection
id.orig_h	addr	Originator's IP address
id.orig_p	port	Originator's port number
id.resp_h	addr	Responder's IP address
id.resp_p	port	Responder's port number
proto	transport_proto	Transport layer protocol of the connection (TCP, UDP, ICMP)
service	string	Application protocol being sent over the connection (HTTP, FTP, DNS, etc)
duration	interval	Duration of the connection
orig_bytes	count	Number of payload bytes the originator sent
resp_bytes	count	Number of payload bytes the responder sent
conn_state	string	State of connection (see Table 2.2)
local_orig	bool	True if connection originated locally or false if originated remotely
missed_bytes	count	Number of bytes missed in content gaps or packet loss
history	string	Records state history of connections as a string of letters (see Table 2.3)
orig_pkts	count	Number of packets sent by originator
orig_ip_bytes	count	Number of IP level bytes sent by originator (taken from IP total_length header field)
resp_pkts	count	Number of packets sent by responder
resp_ip_bytes	count	Number of IP level bytes sent by responder (taken from IP total_length header field)
tunnel_parents	set[string]	Set of uids if connection was over a tunnel

Table 2.2. Bro conn_state Meanings [2]

conn_state	Meaning
S0	Connection attempt seen, no reply.
S1	Connection established, not terminated.
SF	Normal establishment and termination. Note that this is the same symbol as for state S1. You can tell the two apart because for S1 there will not be any byte counts in the summary, while for SF there will be.
REJ	Connection attempt rejected.
S2	Connection established and close attempt by originator seen (but no reply from responder).
S3	Connection established and close attempt by responder seen (but no reply from originator).
RSTO	Connection established, originator aborted (sent a RST).
RSTR	Established, responder aborted.
RSTOS0	Originator sent a SYN followed by a RST, we never saw a SYN ACK from the responder.
RSTRH	Responder sent a SYN ACK followed by a RST, we never saw a SYN from the (purported) originator.
SH	Originator sent a SYN followed by a FIN, we never saw a SYN ACK from the responder (hence the connection was half open).
SHR	Responder sent a SYN ACK followed by a FIN, we never saw a SYN from the originator.
OTH	No SYN seen, just midstream traffic (a partial connection that was not later closed).

nection seen by Bro. Features from the conn.log are used in the machine learning experiments in this thesis, therefore a complete description of it’s contents is warranted. The fields of the default conn.log are presented in Table 2.1. Further explanation of the conn_state and history fields are presented in Tables 2.2 and 2.3, respectively.

2.3 Intrusion Detection Methodologies

The NIST classifies common detection methodologies as signature-based detection, anomaly-based detection, and stateful protocol analysis [40]. Synonymous to signature-based, misuse-detection is common terminology in intrusion detection literature and will be the terminology used in this thesis. All methods can be “stateful”,

Table 2.3. Bro history Meanings [2]

Letter	Meaning
s	a SYN w/o the ACK bit set
h	a SYN+ACK (“handshake”)
a	a pure ACK
d	packet with payload (“data”)
f	packet with FIN bit set
r	packet with RST bit set
c	packet with a bad checksum
i	inconsistent flags in packet

therefore protocol analysis is considered independently. A stateful IDS uses memory to store the state of the environment being observed, allowing state to aid the detection criteria rather than acting on each individual observance alone. For example, a stateful NIDS is capable of detecting threats across multiple packets rather than just one individual packet. This is implemented to assist detection of fragmented attacks, a common IDS evasion technique used by attackers. Hybrid detection methods that combine the use of misuse and anomaly detection are covered in [16]. These four methods are discussed in detail in the following sections.

2.3.1 Misuse Detection.

Misuse detection relies on signatures that correspond to a known vulnerability or threat. A NIDS employing misuse detection will compare network traffic, each individual packet, to a database of signatures producing an alert when there is a match. Systems that use misuse detection are effective at detecting known threats when a corresponding signature is available. Misuse detection is not well suited to detect unknown threats, such as a zero-day attack, or even a variation of known threats. High detection rates with low false positive rates are achievable with a set of well-designed signatures. Snort [5] is the leading open-source misuse detection IDS and is covered in more detail in Section 2.2.1.

2.3.2 Anomaly-Based Detection.

Anomaly-based detection compares observed events to activities considered normal in order to identify significant deviations from normal [40]. A baseline of “normal” must be captured to use as a comparison for the anomaly-based detection system. Challenges in creating a suitable baseline include ensuring that no malicious activity is present and the time period selected to baseline (days, weeks, or months). Activities deviating from the baseline are considered anomalous, but are not always considered a threat.

The primary advantage of an anomaly-based system is the ability to detect unknown threats. One disadvantage is a high rate of false positives, due to the engineering decisions made selecting thresholds for alerts. Another disadvantage is the lack of information provided to an analyst from an alert from an anomaly-based system. In a misuse detection system, analysts can see the signature that caused the alert and compare it to the suspect packet(s). The signature of an anomaly-based system typically relies on statistical methods comparing multiple features that will not be as easy for an analyst to decipher. Anomaly-based alerts typically require more time and expertise to investigate [?].

2.3.3 Stateful Protocol Analysis.

Stateful protocol analysis, sometimes referred to as “deep packet inspection”, compares profiles of acceptable protocol activity for each protocol to observed events to identify deviations [40]. This differs from anomaly-based detection as the focus is on specific protocols rather than network features. Bro [34] is a network security framework that uses protocol analyzers and was covered in detail in Section 2.2.2.

2.3.4 Hybrid Detection.

Hybrid detection methods involve the use of misuse-based and anomaly-based detection with the intent of capitalizing on the benefits of the combined approach. Engineering a solution that maintains the lower false positive rate of misuse detection with the ability to detect unknown threats of anomaly detection could drastically improve the security posture of our networks. Four methods of combining misuse and anomaly detection are misuse-anomaly, anomaly-misuse, parallel, and complex [16, 48, 49]. These hybrid methods are presented in the following sections.

The following sections contains a common use of terms and set notation as presented in [16]. Cyber audit data is represented by set X . A_u and A_n are unknown and normal output from the anomaly detection system, with the properties: $A_u \cup A_n = X, A_u \cap A_n = \emptyset$. M_i and M_u are the intrusive and unknown output from the misuse detection system, with the properties: $M_i \cup M_u = X, M_i \cap M_u = \emptyset$. A detailed description that further breaks each subset into a confusion matrix of true and false positives, and true and false negatives based on known ground truth data can be found in [16, 44].

2.3.4.1 Anomaly-Misuse.

A detection system using the anomaly-misuse serial combination will classify with the anomaly detector first and forward unknown output to the misuse detector. The anomaly-misuse system provides three subsets of X : A_n , $A_u \cap M_i$, and $A_u \cap M_u$, as shown in Figure 2.4, that represent normal, intrusive, and unqualified classes respectively [44]. Intuitively, the intrusive class, $A_u \cap M_i$, contains items that both detectors alerted on and a higher true positive rate is expected for this subset over either detector used independently. Intuitively, the unqualified class, $A_u \cap M_u$, contains items the anomaly and misuse detectors classified as unknown, possibly requiring further

investigation from an analyst. Because A_n is not sent to the misuse detector, this system should have an anomaly detector with a high accuracy rate of the normal class.

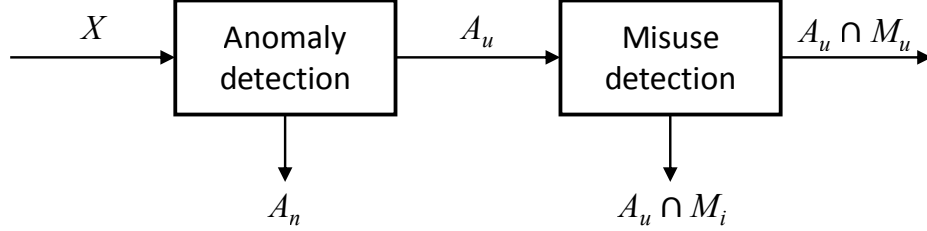


Figure 2.4. Anomaly-Misuse Detection System Workflow [16, 44]

Tombini et al. [44] applied an anomaly-misuse detection system to HTTP traffic resulting in a reduction of 94.25% in log entries (alerts) in the unqualified class when filtering through the anomaly detector compared to using the misuse detector alone. This represents a drastic reduction in unqualified alerts that require investigation from an analyst. Qualitative results are presented showing a combined reduction of 94.52% in unqualified and intrusive classes explaining that a majority of the filtered intrusive class were likely to be false positives from the misuse detector based on the severity levels of the alerts. Ground truth data was unavailable to confirm the qualitative results.

2.3.4.2 Misuse-Anomaly.

A detection system using the misuse-anomaly serial combination will classify with the misuse detector first and forward unknown output to the anomaly detector. The misuse-anomaly system provides three subsets of X : M_i , $M_u \cap A_n$, and $M_u \cap A_u$, as shown in Figure 2.5, that represent intrusive, unknown-normal, and unqualified classes respectively. Intuitively, the unknown-normal class, $M_u \cap A_n$, contains items that the misuse detector classified as unknown and the anomaly detector classified as normal. While still requiring investigation from an analyst, the items in the unknown-normal

class would be a lower priority than the unqualified class, $M_u \cap A_u$, discussed in the previous section.

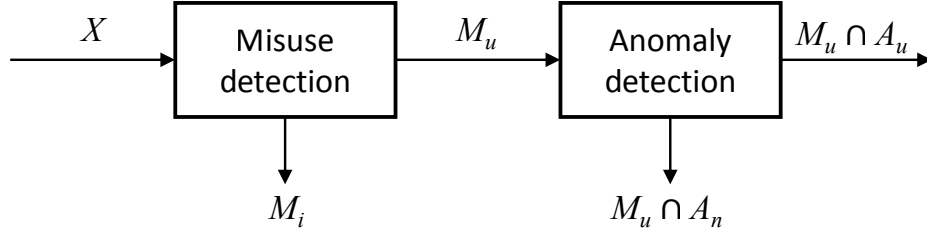


Figure 2.5. Misuse-Anomaly Detection System Workflow [16, 49]

Zhang et al. [48, 49] applied a random-forests-based misuse-anomaly detection system to the KDD'99 dataset resulting in a detection rate of 94.7% with a false positive rate of 2% compared to the misuse detection used independently achieving a detection rate of 94.2% with a false positive rate of 1.1%. The anomaly detection used independently achieved a detection rate of 65% with a false positive rate of 1%. This research concluded that the misuse-anomaly detection system did detect intrusions missed by the misuse approach with minor increase in false positives.

2.3.4.3 Parallel.

A detection system using a parallel configuration will classify all cyber audit data with an anomaly detector and a misuse detector independently. Initially, the outputs A_u , A_n , M_i , and M_u are created as each detector classifies audit data. A correlation function, called `resolver()`, produces subsets $A_n \cap M_i$, $A_n \cap M_u$, $A_u \cap M_i$, and $A_u \cap M_u$ as shown in Figure 2.6. Intuitively, sets $A_n \cap M_u$ and $A_u \cap M_i$ represent instances where both detectors support the other and higher detection rates for normal and intrusive items should be expected. On the other hand, sets $A_n \cap M_i$ and $A_u \cap M_u$ represent instances where the two detectors disagree, requiring an analyst to investigate. An advantage to the parallel configuration is that all subsets of results can be further analyzed compared to how anomaly-misuse does not classify instances in A_n with

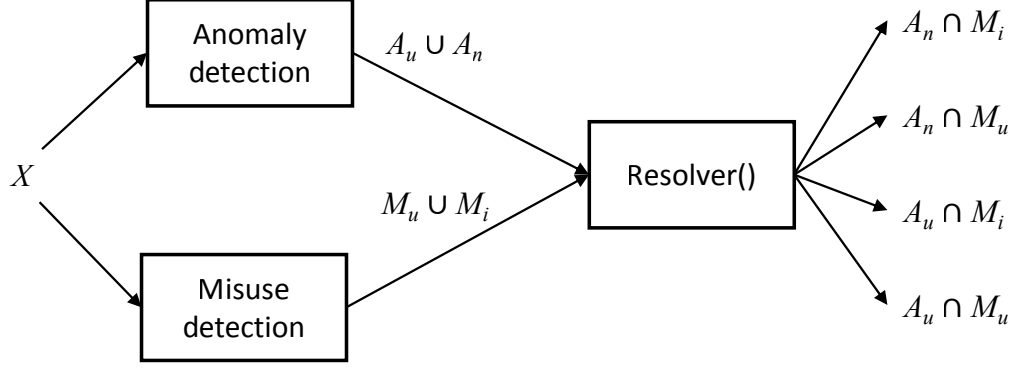


Figure 2.6. Parallel Detection System Workflow [16]

the misuse detector and how misuse-anomaly does not classify M_i with the anomaly detector. Any false negatives in these two subsets have no chance of being discovered by the other detection method. On the other hand, showing all cyber audit data to both detectors could be viewed as a processing disadvantage.

2.3.4.4 Complex.

Complex detection systems use a combination of misuse and anomaly detection but do not exactly fit the structure of anomaly-misuse, misuse-anomaly, or parallel [16]. Examples include the use of multiple machine learning classifiers as the misuse and/or the anomaly detector, producing more combinations of sets and subsets of intrusive, normal, and unknown instances beyond what was presented in the previous sections.

2.4 Machine Learning and Intrusion Detection

Machine learning is a multidisciplinary field that utilizes knowledge from the fields of artificial intelligence, statistics, computational complexity theory, control theory, information theory, philosophy, psychology, neurobiology, and others [29]. Machine learning focuses on learning algorithms that build models from data that can then be used to make decisions or predictions. Thus, machine learning can be considered

a data-driven method of knowledge discovery.

Machine learning algorithms are generally categorized as supervised or unsupervised based on the method used to train the algorithm. In supervised learning, labeled data in the form of instances are supplied to the algorithm to train with. The algorithm builds a model based on the features that correspond to the labeled instances. The labels are also known as classes. In unsupervised learning, the data supplied to the algorithm for training is not labeled. Instead, the unsupervised learning algorithm finds patterns or structure within the data to develop categories, or clusters, that can then be used to make decisions or predictions. More than one type of algorithm can be applied to a problem, for example, a clustering algorithm (unsupervised) can be used to label an unlabeled dataset that a classification algorithm (supervised) then uses to train with. The experiments in this thesis use only supervised learning algorithms, but many types of algorithms are discussed in the following sections.

Machine learning has been applied more generally in the classification of network traffic and more specifically to classifying threats within network traffic. General network traffic classification has many purposes ranging from network planning and provisioning to maintaining Quality of Service (QoS) levels for customers. Network threat detection classification specifically focuses on identifying malicious network traffic. The following sections discuss relevant research in both of these areas.

2.4.1 General Network Traffic Classification.

Traditionally, network traffic was classified based on the set of assigned and ‘well-known’ port numbers. Moore and Papagiannaki [31] demonstrate that recent trends in port assignment and usage result in no better than 70% classification accuracy using the traditional port-based classification methods. Initially, this led to the use of statistical traffic properties and ultimately to the application of machine learning

algorithms to classify network traffic.

In order to generalize network traffic using machine learning, the first step is to extract features to represent the network traffic. Features can be extracted at the packet-level, but are more commonly extracted at the connection-level. Connections, sometimes referred to as flows, consist of one or more packets and represent a session between two hosts for a connection-oriented protocol such as TCP. For connectionless protocols, such as UDP, connections can contain packets to and from the same hosts for a specified duration.

Moore et al. [30] present a comprehensive list of 249 discriminators (features) to use in flow-based classification applications. All features are extracted from packet header information from the packets within a connection. Therefore, each connection represents an instance with 249 features. Statistical features include means, variances, standard deviations, minimums, maximums, and quartile measures for connection-level data such as byte counts, packet inter-arrival times, TCP flag counts, and more.

Nguyen and Armitage [33] present a survey of 17 machine learning research papers focused on the general classification of network traffic. Supervised, unsupervised, and combined approaches are discussed which use various machine learning algorithms and feature sets to conduct experiments on many different datasets. High accuracy (up to 99%) is reported for many offline analysis and a focus for real-time analysis is now trending. A full review of these methods is beyond the scope of this thesis. The intent of this section is to document the related field of general network traffic classification and provide resources to the reader.

2.4.2 Network Threat Detection Classification.

This section discusses the use of machine learning classifiers specific to the application of network threat detection. First, common datasets problems are discussed.

Then a discussion of semantics relating to anomaly detection and threat classification is presented. Finally, a collection of related work is reviewed.

2.4.2.1 Datasets.

Machine learning research in classification of network threats is similar to general network traffic classification and commonly uses similar connection-level features. One difference is requirement for data that contains attacks along with normal traffic. There is a lack of standard datasets or methods to create data to be used in network threat detection research. The commonly used datasets are considered “no longer adequate for any current study” [41], leaving many researchers to produce their own data to conduct experiments that are not easily compared to research performed by others.

A prominent dataset used in IDS research is a synthetic dataset created by MIT’s Lincoln Laboratory with sponsorship by the Defense Advanced Research Project Agency (DARPA) and the Air Force Research Laboratory (AFRL) [28]. Shortcomings of the DARPA ’98 and ’99 datasets and the methods used to create the synthetic network data have been presented by Mahoney and Chan [25] and McHugh [26]. The KDD’99 dataset was derived from the DARPA ’98 dataset for the International Knowledge Discovery and Data Mining competition [17] by processing the raw tcp-dump data into instances of connections represented by 41 features each. While inheriting some of the same criticism as the DARPA datasets, the KDD’99 dataset is the most widely used among IDS researchers. The recently released Kyoto2006+ dataset was created from three years of data (2006-2008) collected from a honeypot at Kyoto University [42]. The Kyoto2006+ dataset attempts to solve the synthetic and out-of-date problems associated with the KDD’99 dataset.

2.4.2.2 Anomaly Detection vs Threat Classification.

It is important to discuss the difference between anomaly detection and other machine learning classification approaches to intrusion detection as the terms are often used interchangeably when, perhaps, they should not be. Symons and Beaver [43] factor the intermixing of terms to the common use of clustering, an unsupervised machine learning approach, to the anomaly detection problem. Normal instances are clustered, then outliers are identified as anomalous.

Sommer and Paxson [41] relate outlier detection of this manner to a “closed world” approach where outliers are assumed to belong to the negative class and consider this as a primary challenge for the application of machine learning to intrusion detection. The authors continue to explain that domains in which machine learning has been applied successfully rely on true classification problems, using samples of positive and negative classes to train a learner. Finally, the authors suggest that machine learning would be better suited for finding *variations of known attacks* rather than discovering *unknown attacks*.

Symons and Beaver [43] argue the semantics of *variations of known attacks* versus *unknown attacks* or *previously unseen attacks* presented by Sommer and Paxson [41], taking the view that normal traffic can be completely different from anything previously seen and that previously unseen attacks may not appear anomalous in the original feature space, but may in fact have distinguishing features that are more similar to known attacks than normal traffic. Their argument is concluded by suggesting the problem is finding the right view, the appropriate expert-derived feature set, through which these distinctions can be made.

This thesis adopts the assumptions of Symons and Beaver [43] on the topic of classifying unknown threats. The supervised algorithms used in the experiments presented in Chapter III are trained with a set of features which the classifier uses to

develop a feature space used to make classification decisions. The developed feature space may resemble that of the original feature space, such as in a decision tree, or it may not resemble the original feature space at all, such as in a neural network. The similarity between a known threat, one which the system has seen, versus an unknown threat, one which the system has not seen, is not simply a variation of the known threat in the original feature set observed by a human analyst, but a similarity in a feature space of the classifier, which was derived from the original feature set by the machine learning algorithm. This similarity in feature space may or may not represent a variation of a known attack or a completely novel attack in terms of the original feature space which a human analyst works with.

There is no common definition of unknown threats with respect to machine learning classifiers and intrusion detection. There is no common method to evaluate a classifier’s ability to detect unknown threats. This property is arguably the most important measure to consider for a classifier used as an IDS and because of this, an experiment and evaluation method is presented in Chapter III in an attempt to address this issue.

2.4.2.3 Related Work.

Beaver et al. [12] developed a system with the objectives of complementing a signature-based IDS, detecting attacks previously unseen by the system, while demonstrating a high detection and low false positive rate. A boosted decision tree algorithm, using Adaptive Boosting (AdaBoost), was tested alongside a signature-based sensor on a self-created dataset comprised of 40 features similar to those from the DARPA 1999 KDD dataset. A penetration test team conducted various attacks, some of which the system was not trained for. The machine learning system detected 82% of the attacks missed by the signature-based system and 89% of the attacks which it

had not been trained. The overall detection rate of the machine learning sensor was 94% with a false positive rate of 1.8%. While the authors demonstrate the usefulness of a hybrid architecture, the complete methodology used to create unknown attacks is not disclosed, leaving the audience to question whether the method of selecting unknown attacks for the experiments was comprehensive enough to validate the results. Unknown attacks with similar features, in the classifier’s feature space, to the attacks which the system was trained could have been used which would lead to high detection rates. The comprehensive unknown threat detection method, such as the one presented in this thesis, would better support such results.

Symons and Beaver [43] address the challenge of training an IDS *in-situ*, in the environment of intended use, by comparing two semi-supervised machine algorithms, Laplacian Eigenmap (LEM) and Laplacian Regularized Least Squares (RLS), to a signature-based system, a linear Support Vector Machine (SVM) classifier, and a maximum entropy classifier. The Kyoto2006+ dataset, which has instances of known and unknown attacks, was used in this research. The authors demonstrate the advantage of the Laplacian RLS algorithm, compared to the other algorithms, when using small training sets. The Laplacian RLS is also shown to perform well when detecting unknown attacks, detecting 397 of the 398 unknown attacks with a false positive rate of 0.01619. The unknown attack detection capabilities of the other machine learning algorithms are not presented in this research. While comparisons are made to a signature-based system, the combined results and value of a hybrid configuration are not discussed in the research.

A summary of the reviewed literature for machine learning methods used in network threat detection is presented in Table 2.4. The algorithm and dataset used is included along with whether a hybrid architecture and unknown threats were considered.

Table 2.4. Summary of Related Work

Author(s)	Algorithm(s)	Dataset	Hybrid	Unk Threats
Zhang et al. [49]	Random Forests	KDD '99	✓	
Beaver et al. [12]	AdaBoost Decision Tree	Self-Created	✓	✓
Symons and Beaver [43]	Laplacian Eigenmap Laplacian RLS Linear SVM Maximum Entropy	Kyoto2006+		✓

2.5 Classifiers

Many machine learning classifier algorithms have been applied to the intrusion detection problem. In the limited scope of this thesis, five algorithms along with two boosted variations will be compared. The Waikato Environment for Knowledge Analysis (WEKA) machine learning suite from the University of Waikato [21] is used to conduct the experiments for this thesis. All of the algorithms used in this research are supervised machine learning algorithms which utilize labeled training data to construct models used to make classification decisions. The following sections will provide the necessary background and relevant WEKA-specific implementations for each of the algorithms used in this research.

2.5.1 Bayesian Network.

Bayesian classifiers are probabilistic and produce probability estimates rather than hard classifications when making classification decisions [46]. Naïve Bayes, a commonly used Bayesian classifier, uses a joint probabilistic model which assumes independence among features. While Naïve Bayes is often successful, assuming independence among features is not always possible, which is when a more general Bayesian Network is preferred. Bayesian Networks are a network of nodes, one node for each feature, connected by directed edges such that there are no cycles, a directed acyclic graph. The connections between nodes represent conditional probabilistic properties

between features. Within each node is a table which defines a probability distribution that is used to produce probability estimates when classifying new instances.

2.5.2 Instance-Based Learner (IBk).

Instance-based learners generate predictions by comparing instances to be classified to the set of saved instances rather than using a set of abstractions of specific instances as a model [9]. The classification decision matches the classification of the closest matching instance in the test set. Instance-based learners are considered lazy algorithms since a model is not built causing the computational expense to occur during classification. IBk is an instance-based learner in WEKA that uses a k -nearest neighbor classifier to determine the closest matching instance with a Euclidean distance function [46].

2.5.3 Sequential Minimal Optimization (SMO).

Sequential Minimal Optimization (SMO) is a training algorithm applied to a Support Vector Machine (SVM) [35]. An SVM algorithm is used for two-class classification problems [13]. Input vectors are non-linearly mapped to a highly dimensional feature space and a linear decision surface is created, separating the two classes. The dimensionality of the feature space is defined by the kernel function selected for use in the SVM algorithm. Linear or polynomial kernel functions can be used. In a two-dimension space, a line would be created to represent the decision barrier while the algorithm seeks to maximize distance from the line for the instances of each class. The instances closest to the decision barrier are called support vectors. In multi-dimension space, the decision barrier is represented by a hyperplane. SVMs can also be used for multi-classification problems by splitting the problem into multiple two-class classification problems and applying an SVM to each subproblem.

The mathematical problem of calculating the optimal hyperplane to use in an SVM is a very large quadratic programming optimization problem [35]. SMO breaks this quadratic problem into a series of the smallest possible quadratic problems, keeping the training process between the linear to quadratic range with regards to the training set size, where the standard SVM algorithm training process falls between the linear to cubic range with regards to the training set size. WEKA utilizes the SMO training process for its SVM implementation.

2.5.4 Decision Trees.

C4.5 is a popular open-source decision tree algorithm developed and improved by Quinlan [37, 36]. A tree structure is produced such that each node represents a decision that corresponds to a specific feature and each leaf node represents a classification decision. Each instance to be classified begins at the root node and traverses down the tree until reaching a leaf node where the classification decision is made. A pruning process is used to reduce the overall size of the decision tree and to reduce the chance of overfitting, a problem that occurs when too many branches are created to represent the training set that ends up not generalizing well with test data.

WEKA's implementation of the C4.5 algorithm is called J48, which implements a later, slightly improved, version of C4.5 before the release of the proprietary version C5.0 [46]. J48 is implemented in Java rather than C, explaining the use of J rather than C in the algorithm's name.

2.5.5 Artificial Neural Networks (ANNs).

Artificial Neural Networks (ANNs) are inspired by biological learning systems comprised of complex networks of interconnected neurons [29]. A set of inputs, the

values of features for each instance, are connected to hidden layers of nodes, which are connected to specific outputs, representing the classes in a classification problem. Each connection is a weighted function that is updated with a training process which maps the input features to the labelled output. The network is trained on multiple iterations of the training data, updating the weight on each iteration until a suitable model is constructed.

WEKA's Multilayer Perceptron (MLP) is an ANN which uses backpropagation for training and allows the user to select the number of hidden layers, training cycles, and other parameters [46].

2.5.6 Boosting Algorithms.

Boosting algorithms are applied to reduce errors of a weak learning algorithm, whose performance is a little better than random guessing [19]. This is accomplished by running the weak learner multiple times over the training data and combining the resulting models from the weak classifier into a combined composite classifier.

WEKA implements the AdaBoost.M1 algorithm which assigns weights to each training instance during each iteration of the boosting algorithm [46]. Instances receive an equal weight for the first iteration, then the weights decrease or increase for correctly and incorrectly classified instances, respectively. This leaves “easy” to classify instances with a low weight and “hard” to classify instances with a high weight. Subsequent iterations of the AdaBoost.M1 algorithm focus on correctly classifying the hard to classify instances, ultimately producing a model with reduced error.

2.6 Summary

In summary, this chapter provided a background of IDS technologies, reviewing the architecture and processes of Snort and Bro. IDS methodologies were discussed in

detail, including the various hybrid IDS configurations. A review of machine learning research applied to general network traffic classification and network threat detection classification was presented. Finally, the necessary background on the classifiers used in the experiments of this thesis was provided.

III. Methodology

3.1 Chapter Overview

This chapter explains the methodology used in this thesis. The first section describes a proposed system architecture which encompasses the methodology used. Next, a description of the dataset used in the experiments is provided. The chapter then discusses the network sensor configuration. The data preprocessing steps are then presented, followed by the classifier configuration. The chapter then explains the machine learning experiments conducted and the performance metrics used for evaluation. Finally, the chapter concludes with the development of a Value-Focused Thinking (VFT) hierarchy to be used for classifier evaluation.

3.2 Proposed System

This research proposes a multi-modality hybrid IDS which uses a packet-level signature-based IDS and a connection-level machine learning threat classifier system. The proposed architecture is provided in Figure 3.1. Network sensor data is classified at the packet-level by the signature-based IDS and also sent to extract connection-level features. In the online process, the connection-level features would be classified by the trained threat classifier(s), then passed to the resolver. The resolver would correlate packet-level alerts to connections, then produce the desired subsets previously discussed in Section 2.3.4 for the analyst to review.

Dua and Du [16] mention the lack of details of the resolver presented by Anderson et al. [10]. A possible implementation in this proposed architecture is to first correlate the packet-based alerts from the signature-based IDS to the connections identified by the connection-level feature extraction, then create a normal and alert set for each classifier, including Snort. Any desired subset combination, for multiple classifiers,

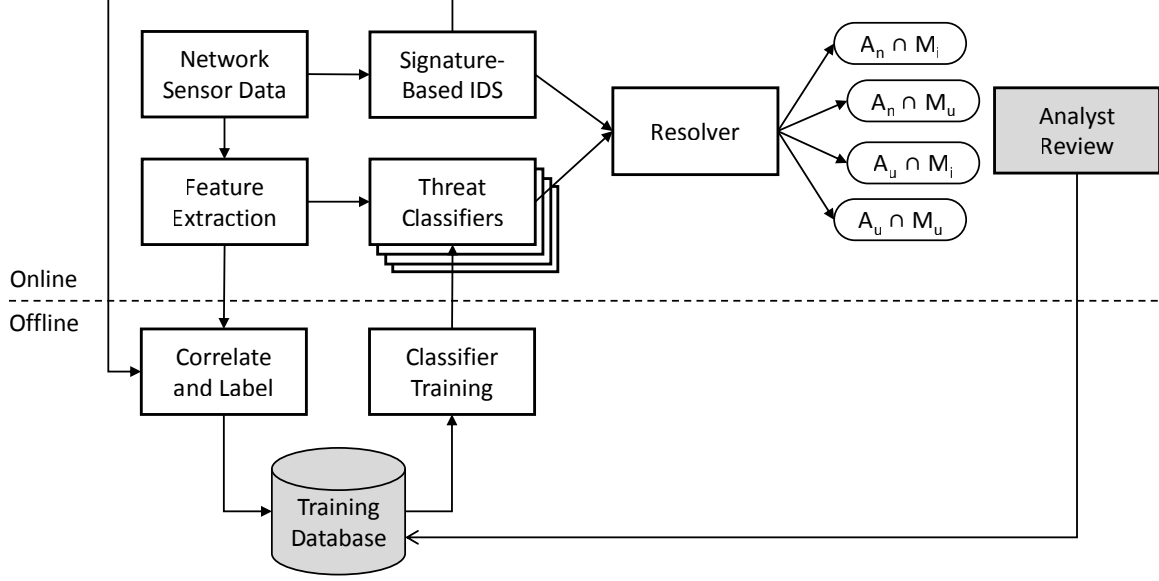


Figure 3.1. Proposed System Architecture

can then be produced from the resolver using simple set operations.

The offline portion illustrates the correlation and labeling process used to create training datasets for the threat classifiers. The shaded components are not addressed in this research, but are provided here as a possible mitigation for the assumption of training classifiers with data that is not labeled with ground truth. The idea being that a human analyst will review and update the training data with ground truth prior to training classifiers. The details specific to each component are covered in the rest of this chapter.

3.3 Dataset Description

As discussed in Chapter II, datasets are among the challenges in IDS research. The data used in this research comes from the Cyber Defense Exercise (CDX), an annual cyberwarfare exercise sponsored by the US National Security Agency (NSA). The network environment and situation in which this dataset originated can be found in [32]. The CDX is an annual competition between military service schools including

the US Military Academy at West Point, US Air Force Academy, Naval Postgraduate School (NPS), US Naval Academy, US Coast Guard Academy, US Merchant Marine Academy, and the Air Force Institute of Technology (AFIT). During the competition, students, as the “blue team”, were tasked to defend similar suites of network equipment and services against attacks. Attacks came from the “red team” which was comprised of highly trained practitioners from the NSA, the Air Force Information Warfare Center, the Navy Information Operations Center, the Army’s 1st Information Operations Command, the Marine Corps Network Operation Center, and the Army Reserve Information Operations Command. The “red team” was able to exploit the “blue team” networks with any open source or publicly available means they desired, leading to a high variation of attack methods from a reputable team of attackers.

The specific data used in this research is the network traffic collected from AFIT’s network during the CDX, as represented in Figure 3.2. All inbound and outbound network traffic was collected in libpcap format using TCPDump. Data was available for years 2003 through 2007, and 2009. Table 3.1 shows the yearly breakout including the total size (in MB) and number of packets collected per year. Having data that spans six years provides the ability to analyze changes in attack methods over time, making it less restrictive than a dataset of a lesser time span.

Table 3.1. CDX Data Set Description by Year

	2003	2004	2005	2006	2007	2009	Total
Size (MB)	1,639	712	704	411	931	502	4,899
Packets	5,861,337	2,888,609	1,609,533	1,130,422	10,452,163	1,808,471	23,750,535

3.4 Network Sensor

The network sensor selected for this research is Security Onion, a Linux distribution that is preloaded with many network security applications. The two primary

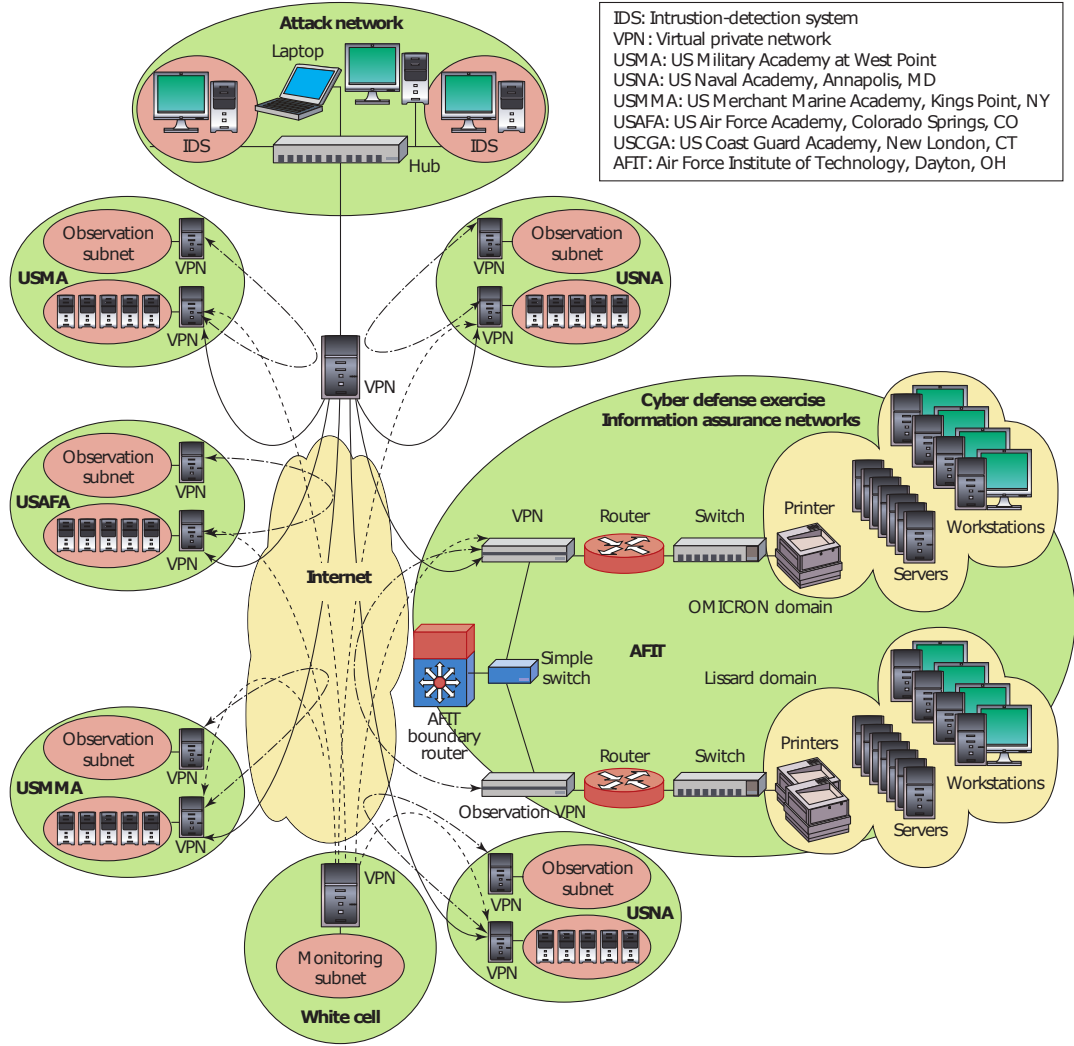


Figure 3.2. CDX Network Architecture [32]

applications used in this research are Bro and Snort. Bro is used to extract connection-level features through the use of the default logs it produces along with additional customized logs. Snort is used to identify packet-level threats. The following briefly explains the configurations of the sensor. A detailed explanation of the configuration process can be found in Appendix B.

The sensor is setup on a virtual machine, such that network traffic can be replayed over the sensor. Security Onion has a built-in setup application for selecting the applications for the sensor and the rule sets to load for the signature-based IDS. Snort is selected as the IDS engine using the “VRT and ET No/GPL” rule set. The total rule count in the set is 21,641. This amount of rules is acceptable for this controlled research experiment, but it is recommended to prune the rule set to 7,000 rules or less in an operational deployment of Security Onion. Because the CDX dataset is from 2003-2009, an assumption is made that the current Snort rule set will identify all of the attacks. Automatic rule updates are disabled to preserve the rule set throughout the experiment.

Table 3.2. Unified2 Log Entry Fields [6]

Field	Description
ts	Timestamp of alert
id.src_ip	Source IP of the packet that generated the event
id.src_p	Source port of the packet that generated the event
id.dst_ip	Destination IP of the packet that generated the event
id.dst_p	Destination port of the packet that generated the event
sensor_id	Unique identifier of sensor
signature_id	Signature ID of the alerting rule, as specified by the sid keyword
signature	Signature message string
generator_id	Generator ID of the alerting rule, as specified by the gid keyword
generator	String description of Snort generator
signature_revision	Revision number of signature
classification_id	Classification ID as mapped in the file classifications.conf
classification	String description of Snort classification
priority_id	Priority of the rule as mapped in the file classifications.conf or overridden by priority keyword
event_id	Unique identifier for each Unified2 event
packet	Raw packet data up to specified byte amount

Snort is configured to output alerts in the Unified2 format, a binary output option which is easily processed into Bro, maintaining consistency between Snort and Bro log files. The fields of the Unified2 alert log entries are shown in Table 3.2. These fields for each alert will be used later for correlating packet-level alerts to connections.

Bro is used to extract connection-level features from the network traffic. In addition to the features in the conn.log described in Table 2.1, Bro is configured to output additional connection-level features shown in Table 3.3. The additional features are derived from the conn.log features. The initial goal was to have Bro output a large set of features as discussed in [30]. Bro’s source code revealed a limitation by recommending the use of connection-level features versus packet-level features. Bro is unable to process events triggered for every packet even at medium level network bandwidths [2].

Table 3.3. Additional Connection Log Features

Feature	Data Type	Description
end_ts	time	Connection finish timestamp
port_service	count	Service identified by port (used when no service detected by protocol analysis)
total_bytes	count	Total number of payload bytes (transport layer)
history (x16)	bool	16 boolean fields added to represent each possible character in conn.log history field
mean_data_ip_orig	count	Mean size (bytes) of packets sent by originator
mean_data_ip_resp	count	Mean size (bytes) of packets sent by responder
total_pkts	count	Total number of packets in connection
total_ip_bytes	count	Total number of IP layer bytes in connection
mean_data_ip	count	Mean size (bytes) of all packets in connection
avg_throughput	double	Average throughput of connection calculated by total_ip_bytes divided by duration

3.5 Data Preprocessing

Data preprocessing includes all necessary actions to get the transform the raw network packet capture data into data ready to be processed by the threat classifiers. The steps include processing the packet capture data through the network sensor, connection-level threat labeling, data cleansing, and data balancing. These steps along with the final dataset descriptions are presented in this section.

3.5.1 Processing Network Packet Capture Data.

Tcpreplay is used to replay the network traffic over the network sniffing interface that was configured during the Security Onion setup. Tcpreplay has options available to control the speed at which packets are transmitted over the selected interface. This is important in the configuration used for this research to ensure that all packets are processed by Snort which is loaded with a large rule set. It was previously mentioned that the recommended size of the rule set is 7,000 or less and this configuration is using 21,641. In order to use the large rule set without packet loss in Snort, packets were replayed at the rate of 100 packets per second. The output of this process can be seen in Figure 3.3. Some of the files processed were removed from the output to

```
rich@rich-security-onion:~$ sudo tcpreplay --pps=100 --intf1=eth1 ~/Desktop/CDX_Data/all_years/*
sending out eth1
processing file: /home/rich/Desktop/CDX_Data/all_years/04-20-2009_wholecapture
processing file: /home/rich/Desktop/CDX_Data/all_years/04-21-2009_wholecapture
<snip>
processing file: /home/rich/Desktop/CDX_Data/all_years/Thursday_2005
processing file: /home/rich/Desktop/CDX_Data/all_years/Wednesday_2005
Actual: 23750535 packets (4527400338 bytes) sent in 262061.68 seconds
Rated: 17276.1 bps, 0.13 Mbps, 90.63 pps
Statistics for network device: eth1
    Attempted packets:      23750535
    Successful packets:     23750535
    Failed packets:         0
    Retried packets (ENOBUFS): 0
    Retried packets (EAGAIN): 0
rich@rich-security-onion:~$
```

Figure 3.3. Example tcpreplay Output

save space, otherwise the results are presented as processed. The total time taken to replay the data at the slower rate was 262,061.68 seconds, approximately 72.8 hours.

The output of tcpreplay indicates that all packets were processed over the interface, but this does not indicate whether or not Bro and Snort was able process each packet. To verify all packets were processed by Snort and Bro, a combination of log files from Security Onion, Snort, and Bro were examined. Once it is confirmed that each application processed all of the data, the log files from each system are retained.

Snort generated 732,709 packet-level alerts from the CDX data, 3.085% of the total number of packets. Figure 3.4 shows the breakout of alerts across 16 of the default alert classifications. Almost half of the alerts reside in the “bad-unknown” class, while “attempted-recon”, “suspicious-filename-detect”, and “suspicious-login”

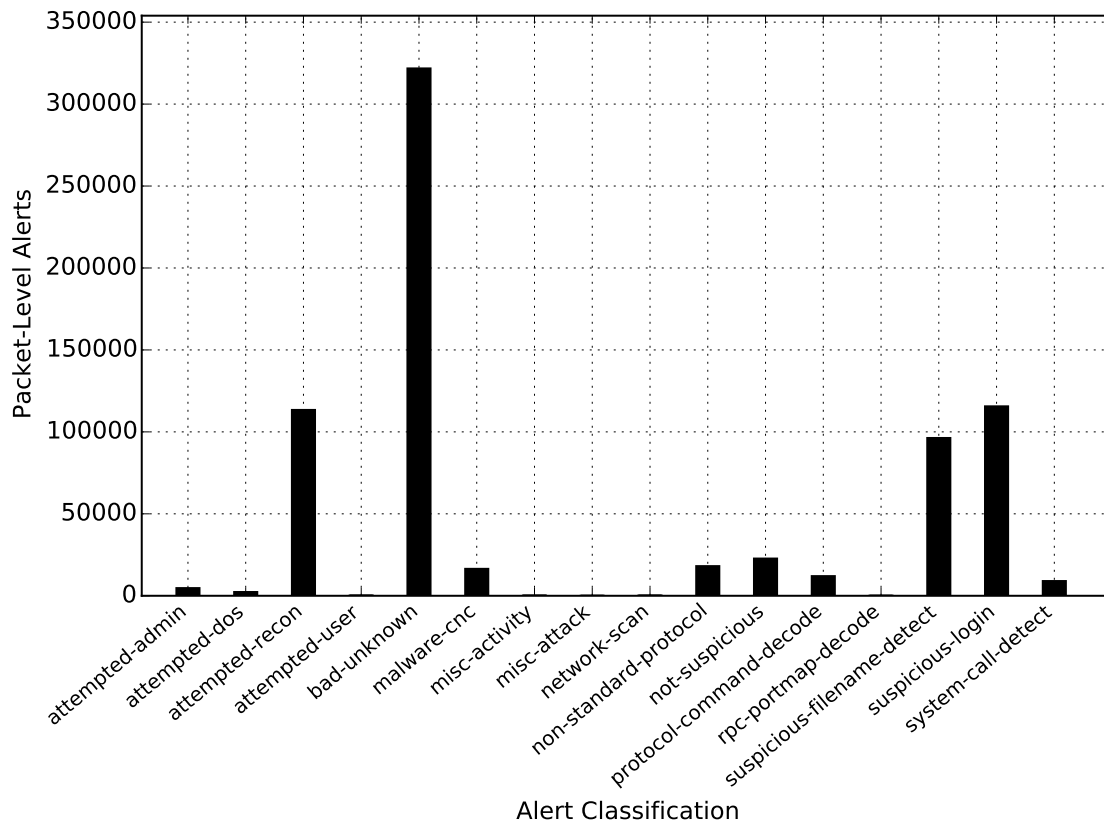


Figure 3.4. CDX Data Packet-Level Alert Classification Metrics

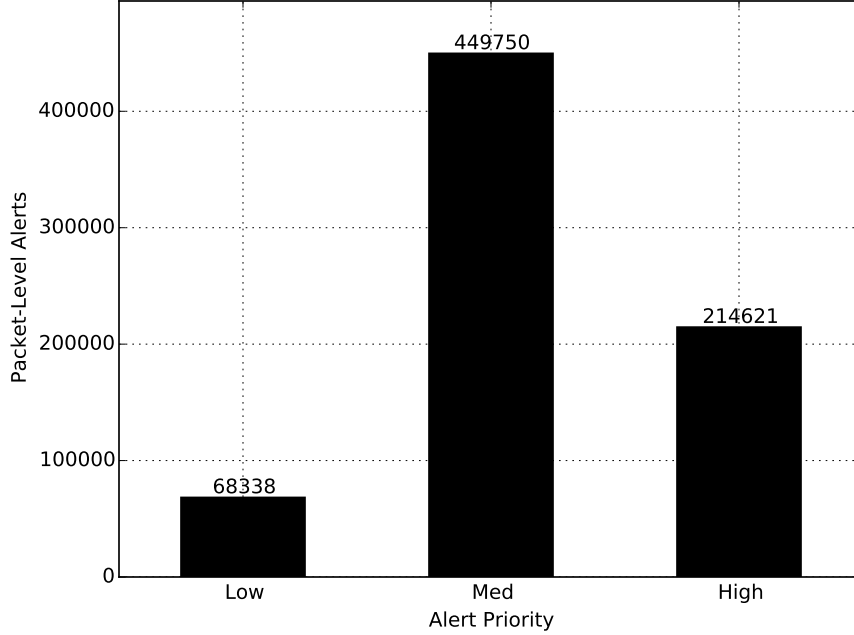


Figure 3.5. CDX Data Packet-Level Alert Priority Metrics

classes are well represented as well. The default alert classifications are mapped to alert priorities in Snort. Figure 3.5 shows a breakout of the alerts by priority from Snort, with the majority of the alerts being medium priority, followed by high, then low priorities. Lastly, there were 169 unique Snort signatures triggered from the CDX data.

Bro generated 3,841,291 connections from the CDX data. Figure 3.6 shows the breakout of connections by service. Services were identified by the service field of the conn.log when available and defaulted to the port.service field when not available. The top ten TCP services are shown along with ICMP and UDP. The tcp.other category represents identified services that were not in the top ten, while the tcp.unknown category represents unidentified services. The high numbers in these categories are attributed to the nature of the dataset and the high number of port scans that were observed.

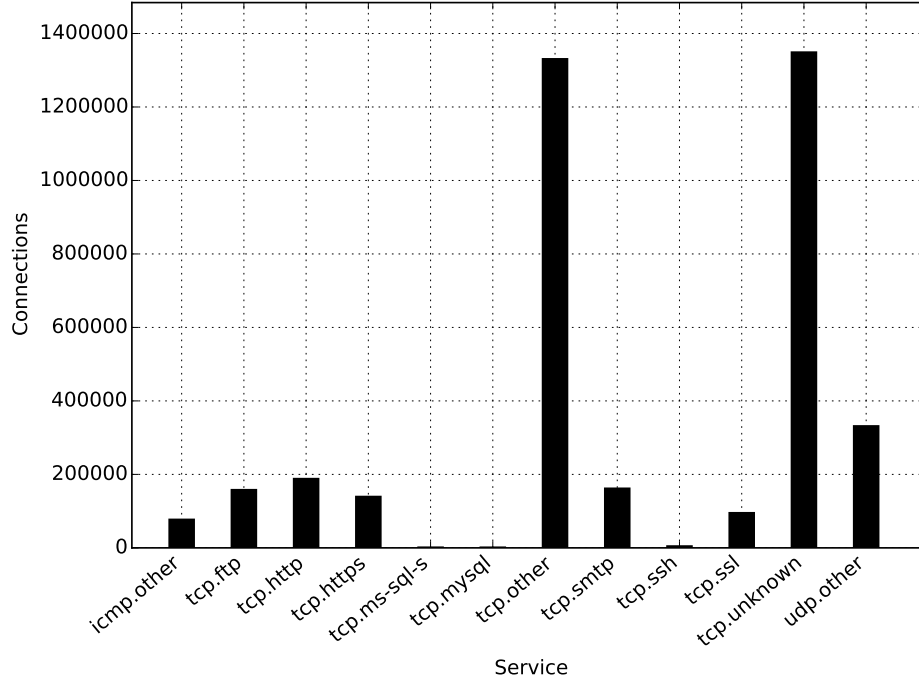


Figure 3.6. CDX Data Connections by Service

3.5.2 Connection-Level Threat Labeling.

Correlating alerts to connections was completed by creating a dictionary containing all alerts, then performing lookups in the dictionary for each connection to check for associated alerts. A dictionary is the Python implementation of hash tables which use key-value pairs to perform fast searches using hashes of the keys, maintaining a performance of $O(1)$ for lookups. Using a hash table data structure the performance of the correlating alerts to connections is $O(n)$, where n is the number of connections. This was valuable when working with large log files of connections and alerts.

The algorithm used to create a dictionary of the alerts in the unified2.log produced by Snort is presented in Figure 3.7. Every alert in the unified2.log is processed through the algorithm and added to the dictionary. In line 2, a frozenset of the network tuple is used as the key. Each connection entry is formatted so the initiating host is the source, while source and destination in Snort alerts are logged as sender and receiver

Input: unified2.log
Output: unified2_dict

```

1: for alert in unified2.log do
2:   key  $\leftarrow$  frozenset(src_ip, src_p, dst_ip, dst_p)
3:   value  $\leftarrow$  [ts, event_id]
4:   if key not in unified2_dict then
5:     unified2_dict[key]  $\leftarrow$   $\emptyset$ 
6:   end if
7:   unified2_dict[key].append(value)
8: end for
9: return unified2_dict

```

Figure 3.7. Alert Dictionary Algorithm

for the packet that caused the alert. This leads to situations where the source and destination in a connection entry are reversed from the Snort alert, but they should be correlated. A set was chosen to allow for this comparison to occur without requiring multiple searches in the dictionary. A frozenset is a Python-specific, hashable version of set that can be used as a key in dictionaries. The value is set in line 3 as a list containing the timestamp of the alert (ts) and the unique event identifier (event_id). Lines 4-6 create an empty list value for the key if it does not already exist in the dictionary, resolving a possible key error. The value is then appended to the list corresponding to the key in line 7. This creates a list of lists for each key if there are more than one alert for a specific key. Finally, the dictionary is returned in line 9.

Once the dictionary of alerts is created, each connection in the conn.features.log can be correlated using the algorithm presented in Figure 3.8. The input to this algorithm is the conn.features.log and unified2.log that were produced through the previously explained Bro configurations. The output will be two new log files, labelled_conn.feature.log, to be used later by the machine learning algorithms, and unified2_conn.log, to document which connections and alerts were correlated allowing for easy cross-referencing from the uid in the connection logs to the event ids in the alert logs when other connection or alert-specific features may be needed. As pre-

Input: conn_features.log, unified2_dict
Output: labelled_conn_features.log, unified2_conn.log

```

1: for line in conn_features.log do
2:   conn_id  $\leftarrow$  frozenset(orig_h, orig_p, resp_h, id.resp_p)
3:   alert_count  $\leftarrow$  0
4:   conn_is_alert  $\leftarrow$  False
5:   if conn_id in unified2_dict then
6:     for alert in unified2_dict[conn_id] do
7:       if ts  $\geq$  (start_ts - 1) and ts  $\leq$  (end_ts + 1) then
8:         alert_count += 1
9:         conn_is_alert  $\leftarrow$  True
10:        write (uid, event_id) to unified2_conn.log
11:        unified2_dict[conn_id].remove(alert)
12:      end if
13:    end for
14:  end if
15:  line.append(alert_count, conn_is_alert)
16:  write line to labelled_conn_features.log
17: end for

```

Figure 3.8. Connection-Alert Correlation and Labeling Algorithm

viously stated, this algorithm is used on every connection in the connection log. In line 2, the conn_id is set to a frozenset of the network tuple, similar to the method used when creating the alert dictionary. This will be used to perform the lookups in that dictionary. In lines 3-4, variables alert_count and conn_is_alert are initialized. The lookup in the alert dictionary occurs in line 5. If there is a match, the conn_id is used to reference the alert dictionary in line 6. Recall that multiple alerts may be saved as a list of lists in each dictionary entry, therefore a for loop is used here check each if they exist. In line 7 a timestamp condition is checked in order to correlate the alert with the connection. The timestamp of the alert (ts) is compared to the connection start timestamp (start_ts) and the connection end timestamp (end_ts) to determine if the alert occurred between the start and end of the connection. A window of 1 second is subtracted from the start timestamp and added to the end timestamp to account for the variation in timestamp precision between the Bro connection log and

the Snort alert log. If the timestamp condition is met, the `alert_count` is incremented, `conn.is_alert` is set to `True`, the `uid` and `event_id` is written to the `unified2.conn.log`, and the alert is removed from the alert dictionary in lines 8-11. The `alert_count` and `conn.is_alert` variables are appended to the connection entry in line 15 and then written to the `labelled_conn.features.log` in line 16.

After completing the correlation and labeling phase, there are 3 log files comprising the entire data set used for the experiments in this research. The `unified2.log`, still in the original form output from the network sensor, contains the alert information which can be referenced individually using the unique event identifiers. The `labeled_conn.features.log`, output from the correlation and labeling process, contains a record of each connection with the features in Tables 2.1 and 3.3 with additional fields representing alert count and boolean representing if the connection is an alert. This labeled dataset is the primer for the data preprocessing phase of the machine learning experiments. Finally, the `unified2.conn.log`, output from the correlation and labeling process, allows easy cross-referencing of between the connection log and the alert log, if needed, using the unique connection and unique alert identifiers, the `uid` and `event_id`.

The connection-alert correlation resulted in 146,531 connections containing one or more packet-level alerts, 3.81% of the total connections. The connection-level threats are shown by service in Figure 3.9. Figure 3.9(a) shows only the connections containing threats while Figure 3.9(b) shows the threat and non-threat connections. Even with a dataset collected in an environment where attacks are imminent, the ratio of threatening connections to non-threatening connections is drastic. It is expected that this difference is amplified in a typical operational network environment, resulting in a larger haystack and smaller needle to search for.

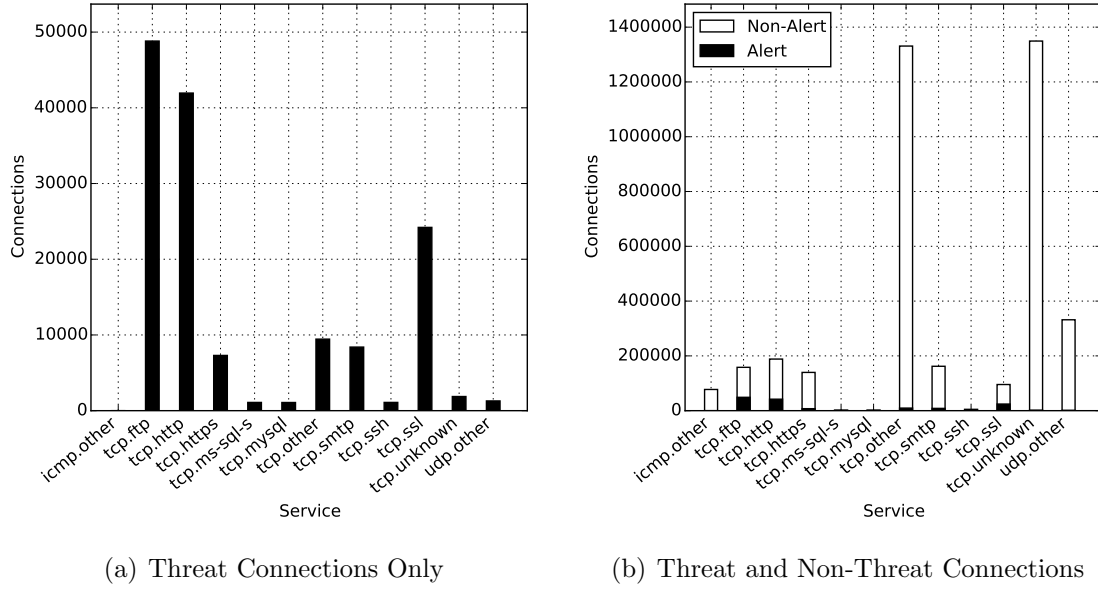


Figure 3.9. CDX Data Connection-Level Threats by Service

3.5.3 Data Cleanup.

Data cleanup, or data cleansing, is typically required to prepare data for use with machine learning algorithms. This step generally involves removing redundant attributes, instances with excessive missing data, and any other changes required to format data specific to the machine learning suite being used. In this case, the data will be reformatted into the WEKA Attribute-Relation File Format (ARFF) and some attributes are removed. The following will explain each attribute that was removed. The `start_ts` and `end_ts` attributes are removed because the data was replayed at a constant speed which is not realistic compared to operational traffic. If temporal relationships were to be considered, the data would need to be replayed in actual time as it was originally transmitted. The `id.orig_h`, `id.orig_p`, `id.resp_h`, and `id.resp_p` were removed to prevent the algorithms from making classification decisions based on the IP and port information. This was decided due to the scenario in which the data was collected. During the CDX, attacks commonly came from the same IP range used by the “red team”. Using this information to make classification decisions

would be unrealistic. All string attributes were removed except for the uid. Those removed were service, port_service, history, and tunnel_parents. It is recognized that using ports and services may be useful for threat classification, but these fields were not used in this research because there were approximately 5,000 unique strings for the service and port_service fields, making it cumbersome to use nominal attributes. Another method of interpreting these string fields could be pursued. As mentioned in Section ??, the history string was parsed into 16 boolean fields to represent the presence of TCP flags. The proto and conn_state attributes were set to be nominal attributes using the sets {tcp, udp, icmp, unknown_transport} and {S0, S1, SF, REJ, S2, S3, RSTR, RSTOS0, RSTRH, SH, SHR, OTH}, respectively. This is one method of preserving string attributes in which the string selections are limited. Instances with missing data were not addressed, but it should be noted that there is an expectation for missing data depending on the protocol of the connection. For example, a UDP connection does not have TCP flags set.

3.5.4 Dataset Subsets.

Subsets of the complete CDX dataset were created containing only the connections specific to each layer, specifically subsets were made for the IP layer, the TCP layer, and application layer, HTTP. The goal of testing separate datasets at different layers of the TCP/IP stack is to test the effect on classifier performance. The hypothesis is that the classifiers will perform better at the higher layers, such as the application layer. For example, a classifier with the sole job of classifying HTTP traffic should perform better than a classifier responsible for classifying all types of TCP or IP traffic.

Separating the dataset by protocol, TCP and IP, was accomplished using the protocol field of the conn.log. The HTTP dataset was created using the service field

or the `port_service` field. The specific condition used is *if* the service field is not empty, *then* service equals the service field, *else* service equals the `port_service` field.

3.5.5 Additional Features.

Datasets with additional features were used for experiments at the application layer. The goal of testing these datasets is to test available features at the higher layers. Introducing these features to the lower layer datasets, TCP and IP, there would be many sparse fields that may affect classifier performance. Performing tests with all available features at the higher layers will further demonstrate whether additional features should be sought if threat classifiers are developed for each application layer. Separate experiments will be conducted on the HTTP dataset with and without the additional features to make comparisons.

The additional features used for the HTTP dataset are derived from the `http.log` produced by Bro. Connections are easily correlated to these logs using the `uid`. All additional features are numeric counts that were extracted from the network traffic by Bro.

Additional HTTP features include counts of HTTP methods, request and response body length statistics (mean, min, and max), server response status codes, and files transferred from the client and server. Only methods and status codes found in the CDX dataset were added as features. Request for Comments (RFC) 7231 [18], recently replacing RFC 2616, provides details on all HTTP methods and server response status codes. A total of 39 additional HTTP features are added to 48,220 connections.

3.5.6 Data Balancing.

Balancing the number of instances from each class is a common tactic used in many machine learning experiments. In this research, imbalanced data was used to

better represent the target environment, where there is much more normal traffic available than there is threat traffic. Implications of learning with imbalanced data are covered in detail in [23]. In this thesis, performance metrics used for evaluation are selected to account for the use of imbalanced data as discussed in Section 3.8.

Random sampling was used to reduce the size of TCP and IP datasets. This was necessary to save computation time. The method used was to retain all threat class instances and use a 10% random sample of the normal class instances. A resampling method included with WEKA was used with a class bias towards uniform distribution of classes (to retain all threat class instances) and without replacement. The specific WEKA method and options used for random sampling can be found in Appendix C.

3.5.7 Final Dataset Descriptions.

The final dataset descriptions are presented in Table 3.4. Each dataset will be processed through each classifier discussed in Section 3.6 under the experiment parameters discussed in Section 3.7.

Table 3.4. Final Dataset Descriptions

Dataset	Connections	Normal	Threat	% Threat	# of Features
All Network Data	384,129	237,598	146,531	38.15	34
TCP Data	343,209	197,965	145,244	42.32	34
HTTP Data	188,371	146,424	41,947	22.27	34
HTTP Data (extra features)	188,371	146,424	41,947	22.27	73

3.6 WEKA Configuration

WEKA version 3.7 was used for this research. All WEKA experiments were performed on a 64-bit Windows 7 SP1 machine with an Intel i7-3720QM CPU (8 cores at 2.6 GHz) and 16 GB of RAM. There are multiple interfaces available in WEKA including a graphical user interface, command line interface, and a Java

Application Program Interface (API) to access the machine learning algorithms in the suite. The Windows command line method was chosen to call on each machine learning algorithm using the Java classes with available options. This allowed the experiments to be scripted with a batch file and get the desired output files to include classifier performance metrics, predictions, threshold information, and details about the classifier models.

Using large datasets with WEKA requires the default memory limit, the maximum heap size, to be increased. This was done by increasing the maxheap value in the RunWeka.ini configuration file in the install directory. The maxheap value dedicates the specified memory amount to the Java virtual machine when WEKA is initialized. A value of 14,336 MB was used for the maxheap in this research, allowing multiple experiments to run simultaneously.

There were 7 classifier types considered in this research for comparison: BayesNet, AdaBoost.BayesNet, IBk, SMO, J48, AdaBoost.J48, and MLP. The specific Java classes called and options used for each classifier can be found in Appendix C.

3.7 Experiments

The experiments for this research were designed for the specific application of threat classifiers used in a hybrid IDS configuration. The ideal threat classifier in this application will have good overall performance with regards to identifying malicious connections versus those that are normal. The classifier should be able to identify known threats as well as unknown threats, while minimizing the amount of false positives and negatives. A false negative equates to a missed detection that may or may not be detected by the misuse detector in a hybrid IDS. A false positive equates to wasted overhead for an analyst reviewing alerts.

It should be noted that it is not possible to test for all unknown threats, as

you cannot evaluate the unknown. We can, however, simulate unknown threats by withholding specific samples from each classifier’s training process.

Two experiments designed to measure classifier performance in these areas are explained in the following sections. The performance metrics discussed in Section 3.8 are used to evaluate each classifier in terms of detecting known and unknown threats while minimizing false positives and negatives.

3.7.1 Classifier Performance.

A stratified k -fold cross-validation experiment is performed on each dataset described in Table 3.4 using each of the classification algorithms. Stratified k -fold cross-validation is a preferred method of testing performance of machine learning algorithms [46]. A select number of folds are created, represented by k . 5-fold and 10-fold are commonly used. In this research, 10-fold is used. Using stratified cross-validation ensures that the distribution of classes, threat and normal in this research, is maintained across each fold.

The goal of the cross-validation experiment is to judge each classifier’s performance in regards to correctly classifying the connections as threat or normal as labeled by the correlation to Snort alerts. Essentially, these experiments rate each classifier’s ability to match the output of Snort. Since Snort uses rules to detect known threats, the results from this experiment equates to measuring the classifier’s ability to detect known threats.

The stratified cross-validation folds used in the experiments were created using a filter provided by WEKA, `weka.filters.supervised.instance.StratifiedRemoveFolds`, where $k = 10$. The specific options used to build the training and test sets are provided in Appendix C.

The training files are used to build a model using each of the tested classifiers.

The model is then used to classify the instances of the corresponding test file. Results are generated for each fold of each experiment.

3.7.2 Unknown Threat Detection.

As previously noted, it is not possible to truly test a classifier’s ability to classify the unknown. We can, however, simulate unknown by selectively withholding samples from the dataset used to train the classifier. While others have presented unknown threat detection metrics [12, 43] in their research, there was not a method used that is as comprehensive as the one used in this thesis.

The unknown threat detection method used here gives every alert classification type within every connection an opportunity to be an unknown threat. The definition of an unknown threat for this experiment is a connection that contains a packet-level alert with an alert classification in which the classifier has not been trained. More specifically, the classifier has not been trained with any connections that contain that specific alert classification.

Training and testing datasets for this experiment are developed similar to the cross-validation method, only instead of have randomly stratified folds, folds are created for each alert classification. The algorithm used for the creation of unknown threat training and testing datasets is presented in Figure 3.10. The inputs are the dataset subset, `unified2.log`, and `unified2_conn.log` (produced from correlation and labeling algorithm in Figure 3.8). The outputs are the testing and training sets for each alert classification for the specified dataset subset where any connection containing the packet-level alert classification being tested for that fold is withheld and all other connections are used for training. Two dictionaries are created, using the same method as Figure 3.7, in the format shown to allow for cross correlation of connections to alert classifications. A `class_set` is created to contain all alert classification types

Input: dataset_subset, unified2.log, unified2_conn.log

Output: dataset_subset_class-train-set, dataset_subset_class-test-set

```

1: for alert in unified2.log do
2:   unified2_dict  $\leftarrow$  (event_id: [classification1, classification2, ..., classificationn])
3: end for
4: for conn in unified2_conn.log do
5:   unified2_conn_dict  $\leftarrow$  (uid: [event_id1, event_id2, ..., event_idn])
6: end for
7: class_set  $\leftarrow$   $\emptyset$ 
8: for line in dataset_subset do
9:   if uid in unified2_conn_dict then
10:    for event_id in unified2_conn_dict[uid] do
11:      class_set.add(unified2_dict[event_id][classification])
12:    end for
13:  end if
14: end for
15: for classification in class_set do
16:   for line in dataset_subset do
17:    if uid in unified2_conn_dict then
18:      temp_set  $\leftarrow$   $\emptyset$ 
19:      for event_id in unified2_conn_dict[uid] do
20:        temp_set.add(unified2_dict[event_id][classification])
21:        if classification in temp_set then
22:          write line to dataset_subset_class-test-set
23:        else
24:          write line to dataset_subset_class-train-set
25:        end if
26:      end for
27:    else
28:      write line to dataset_subset_class-train-set
29:    end if
30:  end for
31: end for

```

Figure 3.10. Unknown Threat Dataset Creation Algorithm

in the dataset subset. Finally, for all classifications in the class_set, correlations are performed in the two dictionaries to select and write the appropriate connections to the testing and training dataset files.

Since the threat classifiers operate at the connection-level and the alert classifications are at the packet-level, it is possible that multiple alert classifications exist

within a connection. Therefore, a single connection may be withheld more than once. Along the same lines, it is also possible that a classifier is trained with a connection that contains a secondary alert classification which exists within a connection being tested. The only way to overcome this issue would be to label each connection with one and only one type of alert classification. This was not pursued in this research. The fact that a connection can contain multiple packet-level alerts is considered a by-product of correlating packet-level alerts to connections and this method accurately describes a malicious connection.

Under these conditions, it is still believed that this experiment measures unknown threat detection capability in a more stringent manner than would occur in an operational environment. Entire alert classification sets are removed together which would not occur in an operational environment. For example, if a novel network scan, an unknown threat, were used in the wild, operational systems would have received training from other types of network scans. In this experiment, the system was not trained on network scans of any type. Therefore, unknown threat detection rates presented using this methodology are expected to be lower than those seen on an operational system. The resulting rate should be a worst-case example, which can then be used to improve upon.

The unknown threat testing and training datasets are subjected to the same classifier configurations used in the classifier performance experiments. Maintaining the same classifier configurations allows for the comparison of classifier performance and unknown threat detection performance of each algorithm.

3.8 Performance Metrics

Many metrics exist to measure the performance of classifiers and should be chosen to best represent the specific classification problem and experiment criteria. It is

uncommon for a single metric to fully capture a classifier’s performance. In this reasearch, attention was taken to select the most appropriate metrics to represent the intended use of threat classifiers in a hybrid IDS architecture where threatening connections are the minority and the cost of false positives and negatives are high. The performance metrics considered in this research are confusion matrices, marginal rates, overall rates, and precision-recall curves. A unique metric, the unknown threat detection rate will be presented as well.

3.8.1 Confusion Matrix.

The metrics used to evaluate the performance of each classifier and fusion methods are derived from the confusion matrix results of each fold of each experiment. Each classifier is making a binary decision for each connection, threat or normal, so a two-class confusion matrix is used. An example of this two-class confusion matrix is presented in Table 3.5. The four values contained within the confusion matrix are defined as:

- *True Positive (TP)*: Instances that are threats which are classified as threats
- *True Negative (TN)*: Instances that are normal which are classified as normal
- *False Positive (FP)*: Instances that are normal, but classified as threats
- *False Negative (FN)*: Instances that are threats, but classified as normal

All confusion matrix values are presented as the sum of the values across the folds of each experiment. These sums represent every connection in each dataset subset.

3.8.2 Marginal Rates.

As previously mentioned, there is no single rate that can accurately depict the performance of a classifier. Marginal rates, the rates derived from the margins of the

Table 3.5. Two-Class Confusion Matrix Example

		Classified as	
		Threat	Normal
Actual	Threat	TP	FN
	Normal	FP	TN

confusion matrix, chosen for evaluation should be selected based on the specific experiment and goals of the evaluator. The metrics derived from the confusion matrices used to compare classifiers and fusion methods are:

- *Recall (True Positive Rate (TPR), sensitivity)*: The rate in which the classifier correctly classifies threat instances. The equation for recall is shown in Equation 3.1.

$$Recall = \frac{TP}{(TP + FN)} \quad (3.1)$$

- *Specificity (True Negative Rate (TNR))*: The rate in which the classifier correctly classifies normal instances. The equation for specificity is shown in Equation 3.2.

$$Specificity = \frac{TN}{(FP + TN)} \quad (3.2)$$

- *Precision*: The rate in which the classifier correctly classifies threat instances to the rate of misclassifying normal instances as threats. The equation for precision is shown in Equation 3.3.

$$Precision = \frac{TP}{(TP + FP)} \quad (3.3)$$

- *False Positive Rate (FPR)*: The rate in which the classifier mis-classifies normal instances as threats to the rate of correctly classifying normal instances. The

equation for FPR is shown in Equation 3.4.

$$FPR = \frac{FP}{(FP + TN)} \quad (3.4)$$

- *False Negative Rate (FNR)*: The rate in which the classifier mis-classifies threat instances as normal to the rate of correctly classifying threat instances. The equation for FNR is shown in Equation 3.5.

$$FNR = \frac{FN}{(TP + FN)} \quad (3.5)$$

All marginal rates are presented as a sample mean, \bar{X} , of the rates from each fold. Equation 3.6 is used to compute sample mean, where r_i is the performance rate for each fold and k is the number of folds. A 95% Confidence Interval (CI) is also provided with each mean rate. Equation 3.7 is used to determine the *CI* of each mean rate, where σ is the standard deviation, n is the sample size, and Z is the z-score (1.96 is used for 95% CI). The sample size ends up being equal to the number of folds.

$$\bar{X} = \frac{r_1 + r_2 + \dots + r_k}{k} \quad (3.6)$$

$$CI = \bar{X} \pm Z \left(\frac{\sigma}{\sqrt{n}} \right) \quad (3.7)$$

3.8.3 Overall Rates.

Overall rates are those rates that attempt to summarize the overall performance of a classifier into one rate. Being that it is difficult to represent classifier performance with one rate, there are many formulas available and each has their pros and cons. The overall rate formulas considered in this research include:

- *Accuracy*: A commonly used rate in which the classifier correctly classifies threat and normal instances to the entire population. Accuracy is a less accurate depiction of overall performance when the data is imbalanced as the majority class will skew the results. For this reason, accuracy is not used as an evaluation metric in this thesis, but is included here as it is commonly used in other research. The equation for accuracy is shown in Equation 3.8.

$$Accuracy = \frac{TP + TN}{(TP + FN + FP + TN)} \quad (3.8)$$

- *F-Measure*: The harmonic mean of precision and recall. Being that it is derived from precision and recall, the TN class is not considered. Making it well-suited for applications using imbalanced data where the correct classification of the negative class is not relevant. F-Measure (Equation 3.9) allows an evaluator to weight precision or recall by adjusting β . The commonly used F_1 -Score balances precision and recall evenly. F_2 weighs recall higher than precision while $F_{0.5}$ weighs precision higher than recall.

$$F_\beta = (1 + \beta^2) \cdot \frac{Precision \cdot Recall}{(\beta^2 \cdot Precision) + Recall} \quad (3.9)$$

- *G-Measure*: The geometric mean of precision and recall. Also derived from precision and recall, the TN class is not considered. Similar to the F-Measure, the G-Measure is sometimes used as a single figure of merit to measure performance in terms of precision and recall. The equation for G-Measure is shown in Equation 3.10.

$$G = \sqrt{Precision \cdot Recall} \quad (3.10)$$

- *Matthews Correlation Coefficient (MCC)*: A measurement that can be applied to a binary classification problems that includes the TN margin. While accuracy includes all margins of a confusion matrix, it is not well suited for imbalanced datasets. The MCC is a measure that includes all margins of the confusion matrix while normalizing the two classes, making it suitable for measuring overall performance with imbalanced datasets. The equation for MCC is shown in Equation 3.11.

$$MCC = \frac{(TP \cdot TN) - (FP \cdot FN)}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \quad (3.11)$$

3.8.4 Precision-Recall Curves.

Precision-recall (PR) curves are used to compare multiple classifiers over the range of precision and recall rates at various threshold settings for each classifier. PR curves are similar to Receiver Operating Characteristic (ROC) curves that use recall and FPR as the axes. Davis and Goadrich [14] explain that PR curves are preferred for use with unbalanced datasets as they provide a more informative picture of an algorithm's performance. The authors continue to show that a curve can only dominate in ROC space if and only if it dominates in PR space. A comparison of two algorithms in ROC space and PR space is presented in Figure 3.11. The optimal area of ROC space is the upper-left corner where $TPR = 1$ and $FPR = 0$. The two algorithms compare well in Figure 3.11(a) and appear to be close to optimal. The optimal area of PR space is the upper-right corner where $Precision = 1$ and $Recall = 1$. When examining the same algorithms in PR space in Figure 3.11(b), Algorithm 2 shows an advantage while it is also evident that both algorithms are less than optimal.

This difference in ROC and PR space occurs in unbalanced datasets because ROC space uses TNs to calculate the FPR. If the majority class is the negative class, there

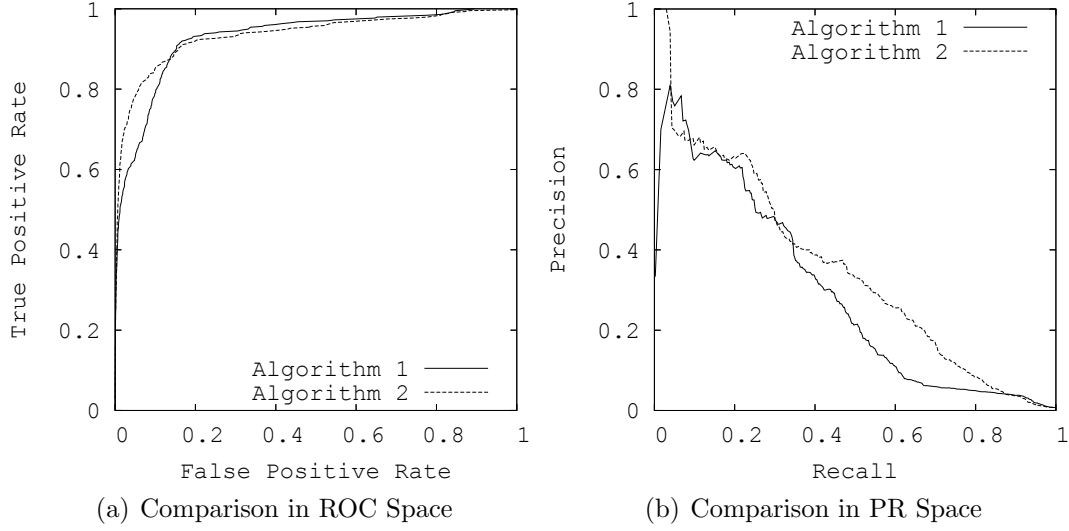


Figure 3.11. Comparing Algorithms in ROC vs PR Space [14]

will be a large amount of TNs. Even a drastic increase in FPs will not have much effect on FPR if there are enough TNs, leading to not much effect to the ROC curve. On the other hand, PR space uses precision, which does not use TNs in the calculation. In this research, normal connections represent the majority class as well as the negative class. Correct classifications of this class is uninteresting to the problem at hand, therefore any metric that includes TNs as part of the calculation, allowing the metric to be skewed by a negative majority class, is considered irrelevant.

To produce a PR curve, precision and recall points must be calculated for varying threshold settings in the classifier. For probabilistic classifiers, the threshold is the probability in which the classification decision is made. Non-probabilistic classifiers producing a discrete classification of 1 or 0 will have a single point rather than a curve. Using the predictions output from WEKA with probabilities, the precision and recall values are calculated for each threshold, producing a curve. PR curves are presented for each classifier performance experiment in this research.

3.8.5 Unknown Threat Detection Rate.

The ability for each classifier and fusion method to detect unknown threats will be measured by the rate in which the classifier correctly classifies the unknown threat TP to entire population of unknown threats in the test sample ($TP + FN$). The equation for the unknown threat detection rate mirrors that of the recall equation presented as Equation 3.1.

Unknown threat detection rates are presented as a measure of *recall* using the sums of TP and FN across all of the alert classification experiments for each classifier. The sum method, rather than using the mean, is used due to the sample size difference in each attack classification experiment. A 95% CI is also provided with each unknown threat detection rate using a binomial distribution. Equation 3.12 is used to determine the confidence interval, CI , of each unknown threat detection rate, where \hat{p} is the proportion, n is the sample size, and Z is the z-score (1.96 is used for 95% CI). The sample size is the total of all test instances, or the sum of all TP s and FN s.

$$CI = \hat{p} \pm Z \sqrt{\frac{1}{n} \hat{p}(1 - \hat{p})} \quad (3.12)$$

3.9 Value Focused Thinking (VFT) Evaluation Approach

Value Focused Thinking (VFT) is a decision-making approach used in operations research domain in which values are weighted in a manner that is relevant to the decision situations of an individual or organization [24]. The basis is that decisions should be made by evaluating values rather than alternatives. While this is not an operational research thesis, a brief explanation of VFT is presented here for completeness.

Values are explicitly weighted by the decision-maker, or evaluator, then constructed in a hierarchical structure of tiers where lower branches represent sub-values

of the parent values. The weighting of the values in each branch of each tier must sum to 1. A function that incorporates the provided weights and performance metrics for each value is then used to compute a score which can be used to evaluate possible decisions.

Current methods used to evaluate machine learning classifiers use combinations of the metrics presented in Section 3.8. The metrics are often used inconsistently across various research experiments. The metrics are not related to organizational values nor do they allow an evaluator to weight metrics according to the organizational values. This research proposes an method of evaluating classifiers based on an organization's values and how the organization weights these values. While organizations may weight values differently from each other, the same organization may weight values differently for different sensors on their network. The use of sensor-specific weighting schemes would allow sensors monitoring a highly sensitive network segment to be weighted differently than a sensor monitoring a less sensitive network segment, selecting an appropriate classification threshold for the specific sensor location. A metric aligned with organizational values also allows for refinement of classifier models toward those values as classifier models can be developed to maximize the metric. The following sections will explain how the VFT approach is applied to evaluating classifiers for use in a hybrid IDS.

3.9.1 Applying VFT to Intrusion Detection.

The proposed VFT method of evaluation is motivated by scenario-based approach to mitigating insider threats [27] which considers benefits and costs associated with implementing security controls to detect insider threat activities. This approach suggests that attack classifications can be substituted in place of scenarios and threat classifiers substituted in place of controls when calculating Figures of Merit (FOM).

A diagram adapted to classifier evaluation by attack classification is presented in Figure 3.12.

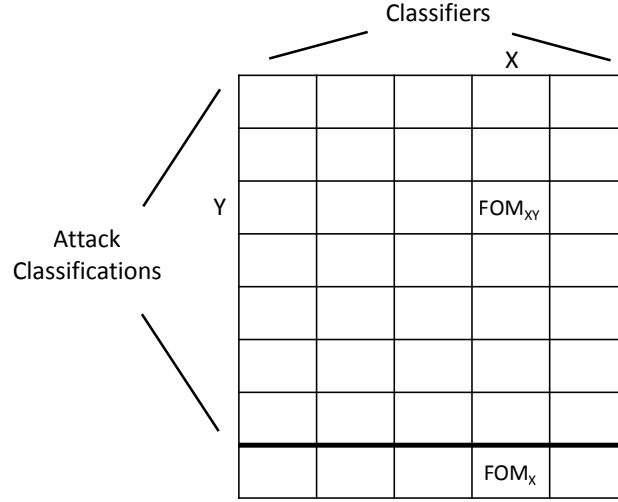


Figure 3.12. Calculating Figures of Merit (adapted from [27])

Mills et al. [27] present a general formula to calculate each FOM using weighted benefits and costs as

$$FOM_{XY} = \sum_j W_{Bj} B_{XYj} - \sum_k W_{Ck} C_{XYk} \quad (3.13)$$

where FOM_{XY} represents a FOM for a specific classifier, X , with regards to a specific attack classification, Y . B_{XYj} and C_{XYk} represent benefits and costs, respectively, associated with each X and Y . Each benefit and cost can be weighted using W_{Bj} and W_{Ck} , respectively. Multiple benefits and costs can be associated with each FOM.

The approach presented in [27] considered binary security controls that were either on or off, with no ability of sensitivity adjustment. The ability to adjust the threshold of a classifier, as discussed in Section 3.8.4 regarding PR and ROC curves, offers a third dimension to the scenario-based approach. Performance metrics can be calculated for each classifier, at each threshold, for each attack classification. A visual depiction of the three dimensions is shown in Figure 3.13.

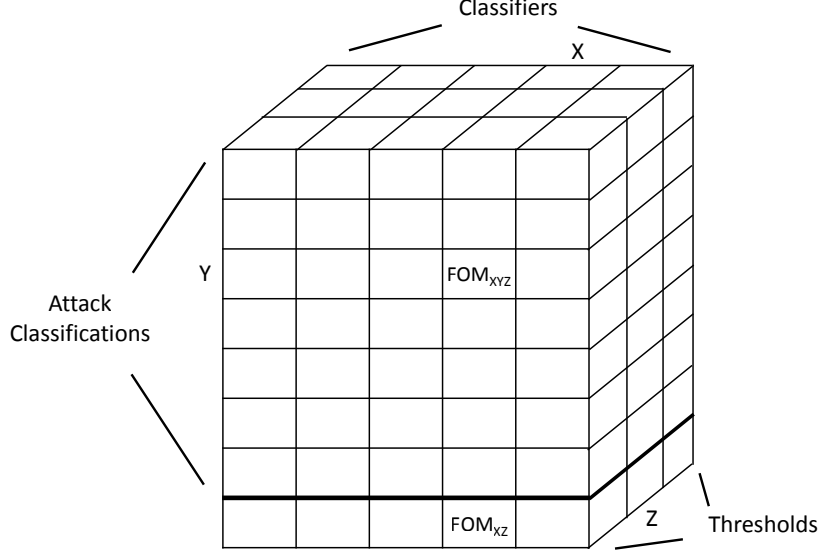


Figure 3.13. Calculating FOM with Classifier Thresholds

VFT comes into play when determining the metrics to use when calculating FOM_{XYZ} and FOM_{XZ} , leading to the proposed value hierarchy in Figure 3.14. The objective of this model is not to select the optimal classifier, but to select the optimal classifier threshold. Optimal, in this situation, is defined as the classifier threshold that receives the highest score based upon the provided value hierarchy with a specific weighting scheme applied. The VFT hierarchy leads to the calculation of each FOM_{XZ} , then the optimal threshold setting is found by the $\max(FOM_{XZ_1}, FOM_{XZ_2}, \dots, FOM_{XZ_n})$, where n is the number of thresholds. The optimal thresholds can then be compared across multiple classifiers to select the optimal classifier and threshold combination, the $\max(FOM_{X_1Z_1^*}, FOM_{X_2Z_2^*}, \dots, FOM_{X_nZ_n^*})$, where n is the number of classifiers and Z_i^* represents the optimal threshold for that classifier X_i .

With the values now defined, the next step is to define the evaluation measures used for each of these values. The following sections explain the evaluation measures for the value hierarchy used to compute FOM_{XYZ} and FOM_{XZ} .

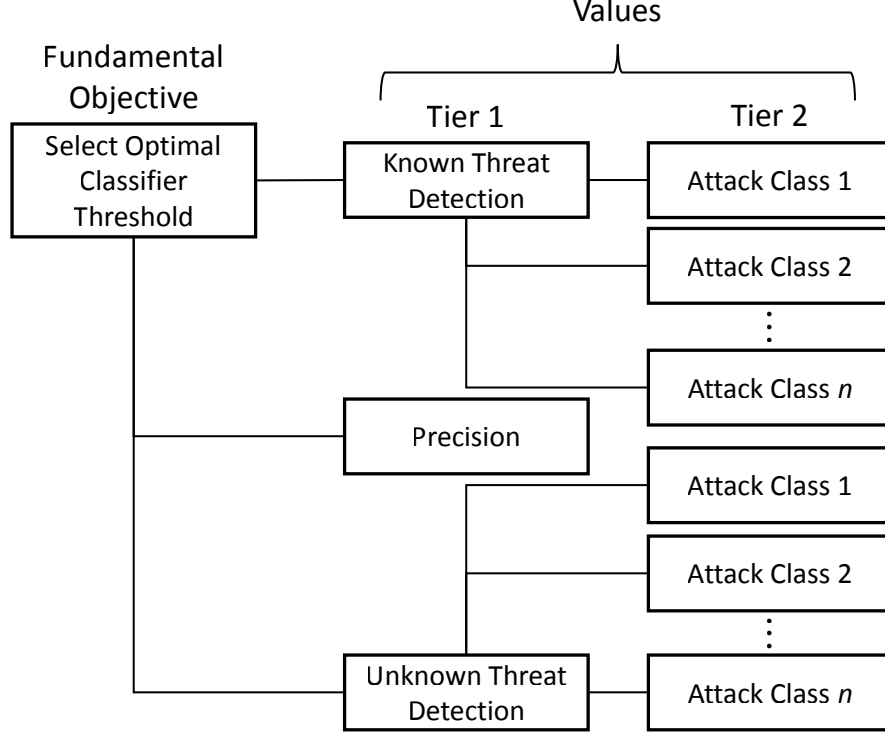


Figure 3.14. VFT Intrusion Detection Hierarchy

3.9.2 Known Threat Detection.

The known threat detection rate (DR_K) represents a classifiers ability to detect malicious connections of a specific attack classification. DR_K is represented by the recall rate from the classifier performance experiments with respect to a specific attack classification category at a specific threshold on the PR curve. Equation 3.14 shows the DR_K calculation as it relates to a specific classifier, attack classification, and threshold, X , Y , and Z respectively.

$$DR_{K_{XYZ}} = Recall_{K_{XYZ}} = \frac{TP_{K_{XYZ}}}{TP_{K_{XYZ}} + FN_{K_{XYZ}}} \quad (3.14)$$

The value of the DR_K is that it represents the ability of the threat classifier to match the classification ability of the packet-level signature-based IDS by using statistical connection-level features with a machine learning algorithm. A high DR_K

will prove higher confidence when comparing alerts from the signature-based IDS and the threat classifier with the intent of supporting true or false positives, the subsets $A_n \cap M_u$ and $A_u \cap M_i$ presented in Section 2.3.4.

A perfect DR_K would result in a threat classifier output that perfectly matches the output of the signature-based IDS. While impressive, it would not add much value over the signature-based IDS alone. A slightly lower DR_K is desired, where the minimal error in DR_K falls into the unknown threat detection metric, explained next.

3.9.3 Unknown Threat Detection.

The unknown threat detection rate (DR_U) represents a classifiers ability to detect malicious connections of a specific attack classification when it has not been trained with malicious examples from said attack classification. DR_U is represented by recall in the unknown threat detection experiments with respect to a specific attack classification category at a specific classifier threshold. Equation 3.15 is similar to the previous equation, except the values for TP and FN are derived from the unknown threat detection experiments.

$$DR_{U_{XYZ}} = Recall_{U_{XYZ}} = \frac{TP_{U_{XYZ}}}{TP_{U_{XYZ}} + FN_{U_{XYZ}}} \quad (3.15)$$

DR_U is arguably the most valuable metric used in the classifier evaluation for a threat classifier used in a hybrid IDS configuration. The notion is that signature-based systems are well-suited to detect known threats, then the primary value of using a threat classifier is to detect unknown threats. A high DR_U will prove higher confidence that the system is capable of detecting threats that are undetected by the signature-based IDS, the subset $M_u \cap A_u$ presented in Section 2.3.4.

Theoretically, a combination of the perfect DR_K and DR_U would result in a threat

classifier output that perfectly matches the output of the signature-based IDS while detecting all unknown threats. While the design of the experiments in this research allow for a DR_K and DR_U of 100%, this would be unachievable in a true operating environment due to the fact that once an unknown threat is detected, the DR_K is no longer 100%. The unknown threat that was correctly detected maintains the perfect DR_U , but since it was not detected by the signature-based IDS, the DR_K will decrease as the threat classifier and signature-based IDS do not perfectly match anymore.

Combining the use of these rates to effectively minimize the subsets of interest in a hybrid IDS configuration, while also detecting all known and unknown threats is the goal. This is subset $M_u \cap A_u$, as presented in Section 2.3.4. The perfect hybrid IDS configuration will have only true unknown threats in this subset.

3.9.4 Precision.

The rate of precision (P) represents the classifiers ability to detect malicious connections while not misclassifying normal connections as malicious. P is represented by the overall precision calculated from the classifier performance experiments. P cannot be calculated per attack classification category as it considers FP metrics that do not specifically relate to an attack class. Equation 3.16 shows the P calculation as it relates to a specific classifier and threshold, X and Z respectively.

$$P_{XZ} = Precision_{XZ} = \frac{TP_{XZ}}{TP_{XZ} + FP_{XZ}} \quad (3.16)$$

Precision is an important metric to include as it expresses the rate in which a FP will occur relative to the amount of TPs. Each connection classified as malicious will require an analyst investigation. Having a high rate of precision results in less FPs per alert shown to an analyst. The ideal rate of precision is 1, where every alert shown to an analyst will be a TP.

3.9.5 VFT Score Calculation.

A complete VFT function used to calculate a score for each classifier threshold, FOM_{XZ} , with a supplied weighting scheme is:

$$FOM_{XZ} = W_{DR_K} \left(\sum_{i=1}^n W_i DR_{K_{XY_iZ}} \right) + W_{DR_U} \left(\sum_{i=1}^n W_i DR_{U_{XY_iZ}} \right) + W_P(P_{XZ}) \quad (3.17)$$

where W_{DR_K} , W_{DR_U} , and W_P represent the tier 1 weights for known threat detection, unknown threat detection, and precision, respectively, and (W_1, W_2, \dots, W_n) , represent the tier 2 weights per attack class, where n is the number of attack classes. Weights must be selected such that the tier 1 weights sum to 1, $W_{DR_K} + W_{DR_U} + W_P = 1$, and the tier 2 weights sum to 1, $\sum(W_1, W_2, \dots, W_n) = 1$.

The optimal classifier threshold, $FOM_{X_iZ}^*$, is then calculated as the maximum of the FOM_{XZ} values for a classifier X_i :

$$FOM_{X_iZ}^* = \max(FOM_{X_iZ_1}, FOM_{X_iZ_2}, \dots, FOM_{X_iZ_n}) \quad (3.18)$$

where n is the number of thresholds for classifier X_i . Each classifier will have an optimal threshold.

Finally, the optimal classifier with threshold, FOM_{XZ}^* , is defined as the classifier with the highest $FOM_{X_iZ}^*$ of the classifiers being evaluated:

$$FOM_{XZ}^* = \max(FOM_{X_1Z}^*, FOM_{X_2Z}^*, \dots, FOM_{X_nZ}^*) \quad (3.19)$$

where n is the number of classifiers. The result is the optimal classifier with the optimal prediction threshold.

3.10 Summary

In summary, this chapter provided the methodology designed to satisfy the research goals of this thesis. First, a proposed system architecture was provided to illustrate how the applications used in this research could be operationally configured. The CDX dataset used was described in detail, followed by the steps taken to configure a network sensor to replay the raw network data then perform the data processing necessary to prepare the data for machine learning experiments. A classifier performance experiment and an unknown threat detection experiment were discussed, along with the evaluation metrics that will be used for each. Finally, the chapter concluded with the development of a VFT evaluation approach tailored to evaluating classifiers for use in hybrid IDS architectures.

IV. Results and Analysis

4.1 Chapter Overview

This chapter presents the results and analysis from the experiments discussed in Chapter III. The first section explores the results from the classifier performance and unknown threat detection experiments evaluated using traditional overall rates, marginal rates, and PR curves. This evaluation is performed on each of the four dataset subsets at varying layers of the TCP/IP stack and with additional features at the application layer. Comparisons are made to determine whether performance is improved by training classifiers to generalize at higher or lower levels of the TCP/IP stack. Next, the VFT evaluation method is applied to the All Network dataset using five notional weighting schemes. The VFT evaluation results are analyzed to determine if there is an engineering advantage when using this method of evaluation over the traditional classifier evaluation methods.

4.2 Classification Results by Dataset

This section compares the performance of various classifiers for each dataset using typical measures for classifier comparisons. The F-Measure, G-Measure, and MCC will be compared as overall performance rates. Precision, recall, and the unknown threat detection rate will be compared as marginal rates. F-Measure, G-Measure, MCC, precision, and recall are presented as mean rates from the withheld test sets from each fold of the 10-fold cross-validation experiment. The unknown threat detection rate is presented as a sum of the withheld test instances from each alert classification in the unknown threat detection experiment. Recall that this research uses imbalanced data sets, therefore metrics that do not account for this, such as accuracy and false positive rates, are not considered.

The methods of comparisons used in this section are often used to select the *optimal* classifier for a specific problem or declare a classifier as the *best* classifier in a research experiment. The “no free lunch” theorem [47] should be considered when labeling classifiers as *optimal* or *best*, with the understanding that the classifier is *optimal* only in the sense of the evaluation criteria, problem, and data used for experimentation. Confusion matrices for each data set showing the results per classifier can be found in Appendix D

4.2.1 All Network Data.

The mean overall rates for the All Network Data dataset are presented in Table 4.1. The J48 and AdaBoost.J48 algorithms consistently score the highest across all three overall rate measurements, followed by AdaBoost.BayesNet, MLP, SMO, and BayesNet in that order. IBk consistently scores the lowest across all three overall rate measurements. Selecting a classifier using these metrics alone would lead to the selection of the J48 or the AdaBoost.J48 algorithms.

Table 4.1. Mean Overall Rates with 95% CI - All Network Data

Classifier	F ₁ -Measure	G-Measure	MCC
AdaBoost.BayesNet	0.9169 (0.9156-0.9181) ●	0.9170 (0.9157-0.9182) ●	0.8671 (0.8652-0.8691) ●
AdaBoost.J48	0.9279 (0.9271-0.9288) ●	0.9281 (0.9272-0.9289) ●	0.8849 (0.8837-0.8862) ●
BayesNet	0.8848 (0.8836-0.8860) ●	0.8848 (0.8836-0.8860) ●	0.8146 (0.8126-0.8165) ●
IBk	0.6841 (0.6237-0.7445) ○	0.7131 (0.6619-0.7643) ○	0.4419 (0.3281-0.5556) ○
J48	0.9274 (0.9266-0.9281) ●	0.9275 (0.9267-0.9283) ●	0.8840 (0.8828-0.8852) ●
MLP	0.9087 (0.9075-0.9099) ●	0.9087 (0.9075-0.9099) ●	0.8526 (0.8505-0.8546) ●
SMO	0.8863 (0.8847-0.8878) ●	0.8863 (0.8848-0.8879) ●	0.8176 (0.8152-0.8200) ●

● Best performer(s) ● Mid-level performer(s) ○ Worst performer(s) statistically per rate

The mean marginal rates for the All Network Data dataset are presented in Table 4.2. AdaBoost.J48 and J48 have leading rates in recall and precision. IBk also has high recall performance, but has the worst precision of all the algorithms, explaining why the overall rates were lower for IBk. The CI is greater on the precision and recall

Table 4.2. Mean Marginal Rates with 95% CI - All Network Data

Classifier	Recall	Precision	UTD Rate
AdaBoost.BayesNet	0.9027 (0.9004-0.9050) ◐	0.9315 (0.9306-0.9323) ◑	0.7327 (0.7306-0.7349) ◐
AdaBoost.J48	0.9132 (0.9118-0.9146) ●	0.9432 (0.9426-0.9438) ●	0.7185 (0.7163-0.7207) ○
BayesNet	0.8791 (0.8775-0.8806) ○	0.8906 (0.8892-0.8920) ◑	0.7766 (0.7746-0.7786) ●
IBk	0.9260 (0.9046-0.9474) ●	0.5614 (0.4570-0.6658) ○	0.7341 (0.7319-0.7362) ◐
J48	0.9129 (0.9115-0.9143) ●	0.9423 (0.9417-0.9429) ●	0.7218 (0.7196-0.7240) ◐
MLP	0.9079 (0.9057-0.9101) ◐	0.9096 (0.9068-0.9124) ◑	0.7160 (0.7138-0.7182) ○
SMO	0.8756 (0.8733-0.8780) ○	0.8971 (0.8958-0.8985) ◑	0.7270 (0.7249-0.7292) ◐

● Best performer(s) ◑ Mid-level performer(s) ○ Worst performer(s) statistically per rate

rates for IBk due to variance across the 10 folds of the cross-validation experiment. When considering the Unknown Threat Detection (UTD) rate, BayesNet performs the best with a mean of 77.66%, while MLP and AdaBoost.J48 perform the worst with means of 71.6% and 71.85% respectively. The mean UTD rate of BayesNet is 4.25% higher than the mean rate of any of the other classifiers (3.84% higher if the 95% CI is considered). By examining marginal rates along with the UTD rate, selecting the “best” classifier becomes more difficult. If the primary goal is to detect unknown threats, BayesNet is the optimal choice, but the ability to match Snort (recall) and maintain the lowest amount of false positives (precision) is the compromise. A mid-level performer, such as AdaBoost.BayesNet, which has slightly lower precision and recall than AdaBoost.J48 and J48, but has a higher UTD rate, might be the preferred classifier. This is a difficult decision to make with these metrics.

The metrics reported in the Tables 4.1 and 4.2 are calculated using the predictions made with the default threshold used by WEKA which makes the classification decision if the prediction probability is greater than or equal to 0.50. The PR curves for the All Network Data dataset presented in Figure 4.1 show the performance of each classifier across many thresholds. The curves for AdaBoost.J48 and J48 dominate the PR space, followed by AdaBoost.BayesNet, then BayesNet.

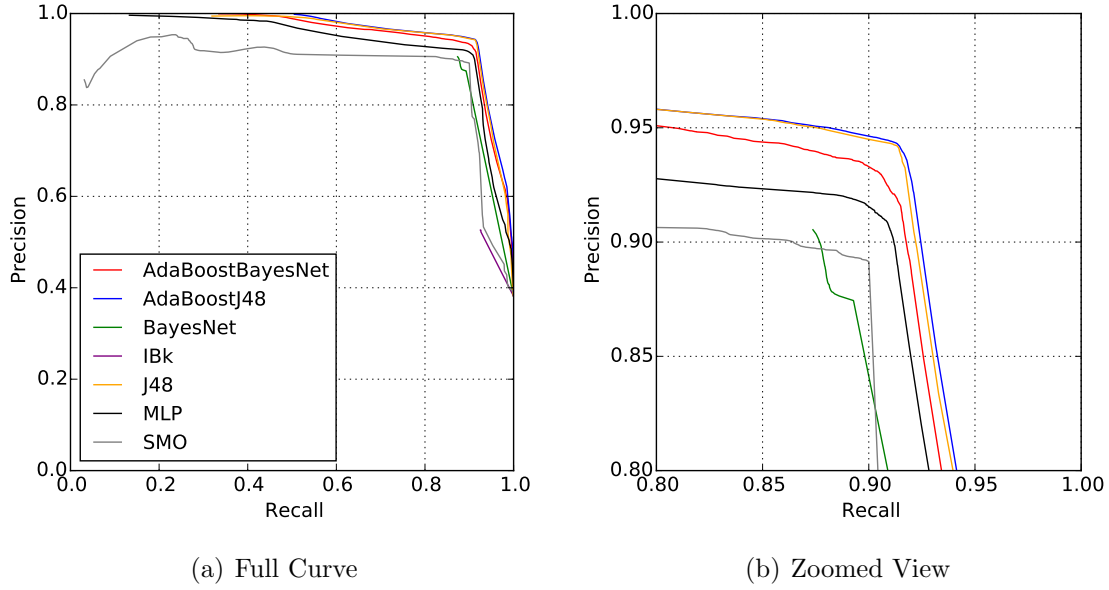


Figure 4.1. Classifier PR Curves - All Network Data

4.2.2 TCP Data.

The mean overall rates for the TCP Data dataset are presented in Table 4.3. Similar to the All Network Data overall rates, the J48 and AdaBoost.J48 algorithms consistently score the highest across all three overall rate measurements, followed by AdaBoost.BayesNet, MLP, SMO, and BayesNet in that order. IBk consistently scores the lowest across all three overall rate measurements. Selecting a classifier using these metrics alone would lead to the selection of the J48 or the AdaBoost.J48 algorithms for the TCP dataset.

Table 4.3. Mean Overall Rates with 95% CI - TCP Data

Classifier	F ₁ -Measure	G-Measure	MCC
AdaBoost.BayesNet	0.9227 (0.9222-0.9233) ●	0.9228 (0.9223-0.9234) ●	0.8678 (0.8669-0.8687) ●
AdaBoost.J48	0.9314 (0.9305-0.9322) ●	0.9315 (0.9307-0.9324) ●	0.8829 (0.8815-0.8844) ●
BayesNet	0.8926 (0.8913-0.8938) ●	0.8926 (0.8914-0.8939) ●	0.8150 (0.8130-0.8171) ●
IBk	0.6857 (0.6455-0.7260) ○	0.7141 (0.6800-0.7482) ○	0.3861 (0.3009-0.4713) ○
J48	0.9307 (0.9298-0.9315) ●	0.9308 (0.9300-0.9316) ●	0.8817 (0.8804-0.8831) ●
MLP	0.9156 (0.9146-0.9167) ●	0.9157 (0.9146-0.9167) ●	0.8544 (0.8526-0.8561) ●
SMO	0.8998 (0.8985-0.9010) ●	0.8998 (0.8985-0.9011) ●	0.8272 (0.8250-0.8293) ●

● Best performer(s) ● Mid-level performer(s) ○ Worst performer(s) statistically per rate

Table 4.4. Mean Marginal Rates with 95% CI - TCP Data

Classifier	Recall	Precision	UTD Rate
AdaBoost.BayesNet	0.9093 (0.9077-0.9109) ◐	0.9366 (0.9352-0.9381) ◑	0.7424 (0.7402-0.7445) ◐
AdaBoost.J48	0.9153 (0.9139-0.9167) ◐	0.9480 (0.9470-0.9490) ●	0.7283 (0.7261-0.7305) ◐
BayesNet	0.8849 (0.8830-0.8869) ○	0.9004 (0.8991-0.9017) ◑	0.7808 (0.7788-0.7829) ●
IBk	0.9385 (0.9221-0.9548) ●	0.5492 (0.4776-0.6208) ○	0.7413 (0.7391-0.7434) ◐
J48	0.9146 (0.9132-0.9161) ◐	0.9473 (0.9463-0.9482) ●	0.7285 (0.7263-0.7307) ◐
MLP	0.9108 (0.9090-0.9127) ◐	0.9205 (0.9194-0.9216) ◑	0.6998 (0.6975-0.7020) ○
SMO	0.8935 (0.8916-0.8953) ◐	0.9061 (0.9049-0.9074) ◑	0.7412 (0.7390-0.7433) ◐

● Best performer(s) ◐ Mid-level performer(s) ○ Worst performer(s) statistically per rate

The mean marginal rates for the TCP Data dataset are presented in Table 4.4. *IBk* has a slight statistical advantage over the other classifiers in terms of recall, but this gain is attributed to a low rate of precision. AdaBoost.J48 and J48 follow closely in terms of recall and have the statistical advantage in terms of precision. BayesNet has the lowest rate of recall at 88.49%, but has the highest UTD rate at 78.08%, which is 3.84% higher than the mean rate of any of the other classifiers (3.43% higher if the 95% CI is considered). Again, it is difficult to make a classifier selection by comparing precision, recall, and UTD rates. Depending on the desired application, an evaluator could choose AdaBoost.J48 or J48 based on recall and precision performance, BayeNet based on UTD performance, or a mid-level performer that better balances each rate.

The PR curves for the TCP Data dataset are presented in Figure 4.2. The curves are very similar to those from the All Network Data dataset, with AdaBoost.J48 and J48 dominating the PR space, followed by AdaBoost.BayesNet, then BayesNet.

To compare the classification performance between the All Network and TCP datasets, the TCP connections were extracted from the classifier performance and unknown threat detection experiments performed with the All Network dataset and the marginal rates for recall and UTD rate were calculated. Due to uneven sampling of TCP connections per fold in the All Network dataset, these rates are calculated using the total number of instances classified from the positive class (threat connections)

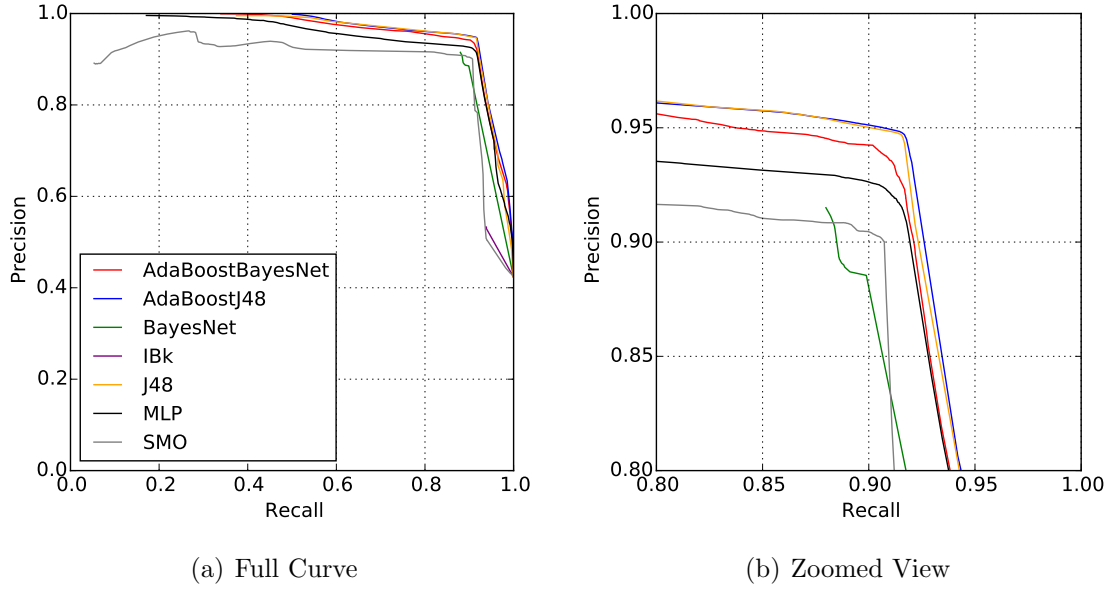


Figure 4.2. Classifier PR Curves - TCP Data

across the 10 cross-validation folds, rather than the mean rates of the 10 folds. The overall rates and precision were not compared as they involve calculations using the negative class (normal connections) which vary among the All Network and TCP datasets due to the random sampling process described in Section 3.5.6. These results are presented in Table 4.5. There was not a significant difference in performance between the All Network and TCP datasets in terms of recall and UTD rates. This

Table 4.5. Performance Comparison of TCP-Only Connections

Classifier	All Network		TCP	
	Recall	UTDR	Recall	UTDR
AdaBoost.BayesNet	0.905497	0.734456	0.909297	0.742404
AdaBoost.J48	0.914792	0.720397	0.915349	0.728314
BayesNet	0.884085	0.780743	0.884911	0.780843
IBk	0.926951	0.735656	0.938455	0.741292
J48	0.914723	0.723796	0.914606	0.728520
MLP	0.914613	0.721503	0.910840	0.699791
SMO	0.883300	0.733156	0.893469	0.741185

is as expected due to the difference of only 1,287 of the positive class between the two datasets, 146,531 in the All Network dataset compared to 145,244 in the TCP dataset. If comparisons could be made among the negative class in each dataset, a difference in precision may be present.

4.2.3 HTTP Data.

The mean overall rates for the HTTP Data dataset are presented in Table 4.6. J48 and AdaBoost.J48 algorithms consistently score the highest across all three overall rate measurements, followed by AdaBoost.BayesNet, IBk, MLP, and BayesNet in that order. SMO consistently performed the worst across all three overall rate measurements.

Performance from SMO was significantly degraded compared to the SMO performance in the All Network and TCP datasets, with an average decrease in mean rates of 8.38% in the All Network dataset and 9.6% in the TCP dataset across all three overall rates. Performance from IBk was significantly improved compared to the IBk performance in the All Network and TCP datasets, with an average increase in mean rates of 32.94% in the All Network dataset and 34.71% in the TCP dataset across all three overall rates. These changes in performance indicate that IBk may have improved performance at higher levels of the IP stack while SMO may have improved

Table 4.6. Mean Overall Rates with 95% CI - HTTP Data

Classifier	F ₁ -Measure	G-Measure	MCC
AdaBoost.BayesNet	0.9521 (0.9503-0.9539) ●	0.9521 (0.9503-0.9540) ●	0.9386 (0.9363-0.9409) ●
AdaBoost.J48	0.9664 (0.9651-0.9676) ●	0.9664 (0.9652-0.9676) ●	0.9568 (0.9552-0.9583) ●
BayesNet	0.8574 (0.8558-0.8591) ●	0.8650 (0.8636-0.8664) ●	0.8223 (0.8203-0.8242) ●
IBk	0.9474 (0.9463-0.9486) ●	0.9475 (0.9463-0.9486) ●	0.9324 (0.9308-0.9339) ●
J48	0.9681 (0.9673-0.9689) ●	0.9682 (0.9674-0.9690) ●	0.9592 (0.9582-0.9602) ●
MLP	0.9069 (0.8989-0.9148) ●	0.9078 (0.8998-0.9158) ●	0.8827 (0.8718-0.8936) ●
SMO	0.7981 (0.7950-0.8011) ○	0.7983 (0.7954-0.8013) ○	0.7424 (0.7389-0.7459) ○

● Best performer(s) ● Mid-level performer(s) ○ Worst performer(s) statistically per rate

Table 4.7. Mean Marginal Rates with 95% CI - HTTP Data

Classifier	Recall	Precision	UTD Rate
AdaBoost.BayesNet	0.9436 (0.9410-0.9463) ●	0.9607 (0.9581-0.9633) ●	0.5574 (0.5530-0.5619) ●
AdaBoost.J48	0.9623 (0.9604-0.9642) ●	0.9704 (0.9693-0.9715) ●	0.5581 (0.5536-0.5625) ●
BayesNet	0.9876 (0.9861-0.9892) ●	0.7576 (0.7546-0.7606) ○	0.6086 (0.6043-0.6130) ●
IBk	0.9479 (0.9463-0.9495) ●	0.9470 (0.9451-0.9489) ●	0.4634 (0.4590-0.4679) ●
J48	0.9584 (0.9565-0.9602) ●	0.9781 (0.9769-0.9793) ●	0.5485 (0.5441-0.5530) ●
MLP	0.8792 (0.8641-0.8942) ●	0.9384 (0.9125-0.9644) ●	0.2871 (0.2830-0.2911) ○
SMO	0.7794 (0.7733-0.7855) ○	0.8177 (0.8163-0.8191) ●	0.2882 (0.2841-0.2922) ○

● Best performer(s) ● Mid-level performer(s) ○ Worst performer(s) statistically per rate

performance at the lower levels of the IP stack, but this could be attributed to the classification performance of HTTP traffic alone. This is examined further in the following section.

The mean marginal rates for the HTTP Data dataset are presented in Table 4.7. BayesNet is the leader in terms of recall by difference in means of 2.53% (2.19% if the 95% CI is considered) and the leader in terms of UTD rate by a difference in means of 5.05% (4.18% if the 95% CI is considered). The high performance in recall and unknown threat detection come at the price of a reduced level of precision, the lowest mean precision rate of the group at 75.76%. AdaBoost.BayesNet, AdaBoost.J48, J48, and IBk all perform well in terms of recall and precision, with rates ranging from 94.36-96.23% for recall and 94.7-96.07% for precision. The mean UTD rates for AdaBoost.BayesNet, AdaBoost.J48, and J48 range from 54.85-55.81%, while IBk is significantly lower at 46.34%. The UTD rates are significantly lower in the HTTP dataset than the All Network or TCP datasets, by the largest difference of 45.3% for SMO from the TCP dataset to the HTTP dataset and by smallest difference of 16.04% for AdaBoost.J48 from the All Network dataset to the HTTP dataset. This could be attributed to the possibility that variations of HTTP threats are more difficult to classify or that their are non-HTTP threats in the All Network and TCP datasets in which the variations are easier to classify, in turn raising the UTD rates. This is

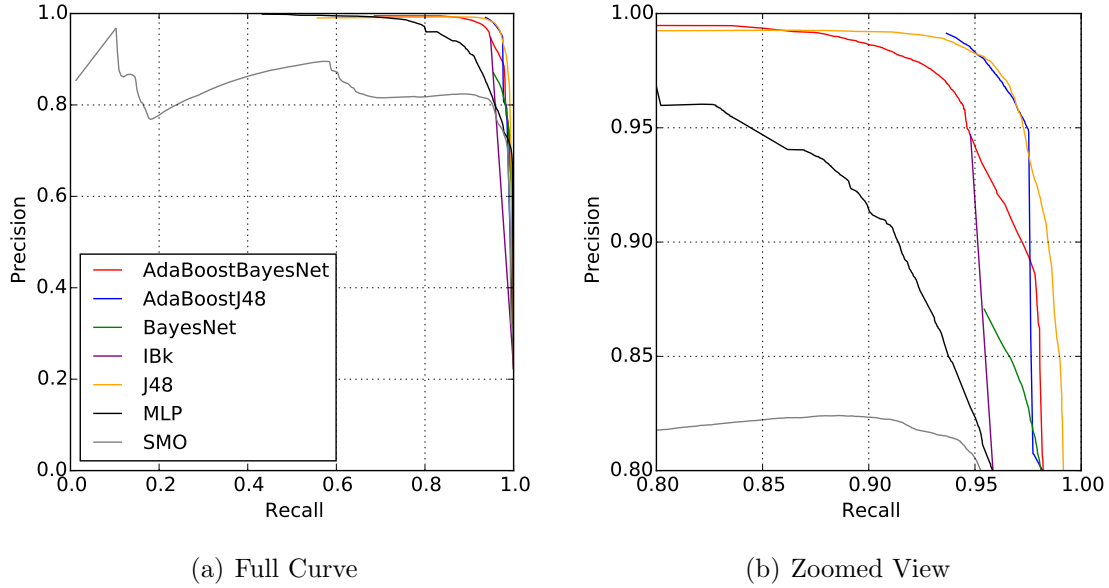


Figure 4.3. Classifier PR Curves - HTTP Data

examined later by extracting the HTTP threat connections from the All Network and TCP datasets and making comparisons to the results from the HTTP dataset.

The PR curves for the HTTP Data dataset are presented in Figure 4.3. The curves for AdaBoost.J48 and J48 continue to dominate the PR space with the HTTP dataset as they did with the All Network and the TCP datasets. AdaBoost.BayesNet is also a strong performer.

4.2.4 HTTP Data with Additional Features.

The purpose of performing experiments on the HTTP dataset with additional features is to determine whether other features derived from Bro's http.log can improve classifier performance. Rather than make comparisons among the various classifiers, each rate is marked whether or not there was a statistically significant change in comparison to the rates from the HTTP dataset without the additional features. The mean overall rates for the HTTP Data with Additional Features dataset are presented in Table 4.8. There was a consistent improvement across the mean overall

Table 4.8. Mean Overall Rates with 95% CI - HTTP Data with Additional Features

Classifier	F ₁ -Measure	G-Measure	MCC
AdaBoost.BayesNet	0.9536 (0.9520-0.9551) –	0.9536 (0.9521-0.9551) –	0.9405 (0.9386-0.9424) –
AdaBoost.J48	0.9691 (0.9676-0.9706) ↑	0.9691 (0.9676-0.9706) ↑	0.9603 (0.9584-0.9622) ↑
BayesNet	0.8588 (0.8573-0.8602) –	0.8661 (0.8647-0.8674) –	0.8238 (0.8220-0.8256) –
IBk	0.5255 (0.4651-0.5860) ↓	0.5801 (0.5195-0.6406) ↓	0.3877 (0.2903-0.4850) ↓
J48	0.9705 (0.9692-0.9719) ↑	0.9706 (0.9692-0.9720) ↑	0.9623 (0.9606-0.9641) ↑
MLP	0.8086 (0.7546-0.8626) ↓	0.8217 (0.7749-0.8685) ↓	0.7831 (0.7307-0.8356) ↓
SMO	0.9133 (0.9121-0.9146) ↑	0.9134 (0.9121-0.9147) ↑	0.8889 (0.8872-0.8906) ↑

↑ Statistically significant improvement ↓ Statistically significant degradation – No statistically significant change

rates for AdaBoost.J48, J48, and SMO, consistent degradation for IBk and MLP, and no significant change for AdaBoost.BayesNet and BayesNet.

The mean marginal rates for the HTTP Data with Additional Features dataset are presented in Table 4.9. AdaBoost.J48 and J48 saw improvements in recall and/or precision with no significant change in UTD rate. SMO saw improvements in all rates. This suggests that J48 and SMO both responded positively to the additional features used. AdaBoost.BayesNet and BayesNet saw no significant change in precision or recall, but did see a degradation in UTD Rate. MLP saw a degradation in recall and UTD rate and IBk saw a drastic degradation in precision. This suggests BayeNet, MLP, and IBk all responded negatively to the additional features used.

The PR curves for the HTTP Data with Additional Features dataset are presented in Figure 4.4. The PR curves are similar to the HTTP dataset with the use of the

Table 4.9. Mean Marginal Rates with 95% CI - HTTP Data with Additional Features

Classifier	Recall	Precision	UTD Rate
AdaBoost.BayesNet	0.9441 (0.9407-0.9474) –	0.9633 (0.9619-0.9646) –	0.5124 (0.5080-0.5169) ↓
AdaBoost.J48	0.9650 (0.9630-0.9670) –	0.9732 (0.9717-0.9748) ↑	0.5589 (0.5545-0.5634) –
BayesNet	0.9868 (0.9850-0.9885) –	0.7602 (0.7577-0.7626) –	0.5847 (0.5803-0.5891) ↓
IBk	0.9093 (0.8227-0.9959) –	0.3724 (0.3201-0.4248) ↓	0.9038 (0.9012-0.9065) ↑
J48	0.9602 (0.9584-0.9621) ↑	0.9811 (0.9794-0.9827) ↑	0.5533 (0.5488-0.5577) –
MLP	0.7400 (0.6351-0.8449) ↓	0.9304 (0.8836-0.9771) –	0.2561 (0.2522-0.2600) ↓
SMO	0.9045 (0.9025-0.9066) ↑	0.9223 (0.9198-0.9249) ↑	0.5207 (0.5162-0.5252) ↑

↑ Statistically significant improvement ↓ Statistically significant degradation – No statistically significant change

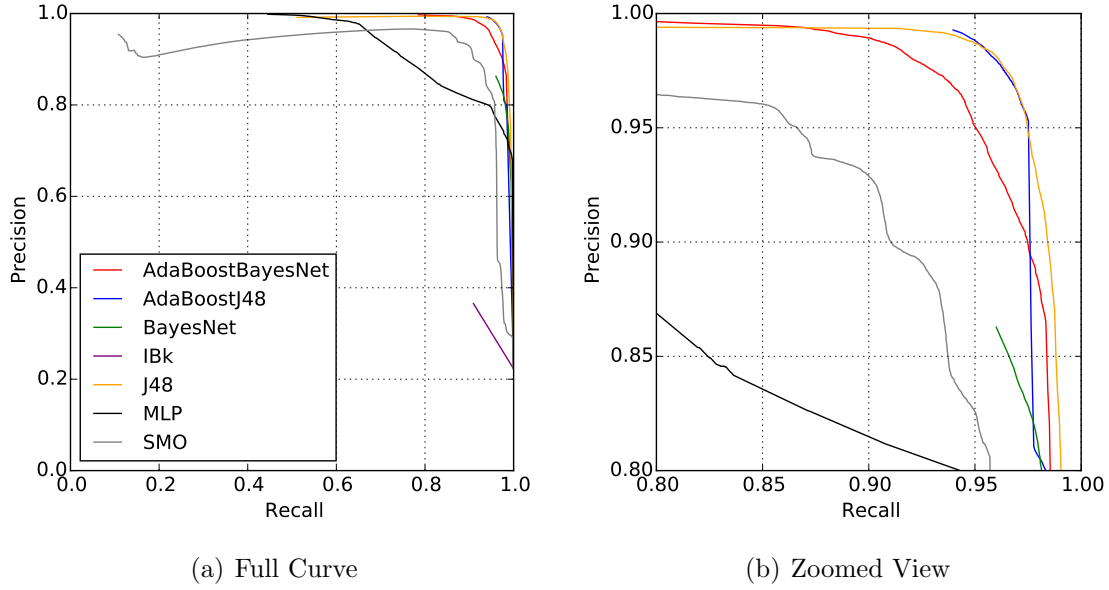


Figure 4.4. Classifier PR Curves - HTTP Data with Additional Features

additional features. An improvement for SMO and a degradation for MLP can be noticed across the threshold ranges.

To compare the classification performance between the All Network, TCP, and HTTP datasets, the HTTP connections were extracted from the classifier performance and unknown threat detection experiments performed with the All Network and TCP datasets and the marginal rates for recall and UTD rate were calculated. Similar to the TCP connection comparison, precision and the overall rates were not compared as the calculations include the negative class (normal connections) which vary among the All Network and TCP datasets due to the random sampling process described in Section 3.5.6.

The results presented in Table 4.10 show a degradation in performance at the HTTP layer when compared to the All Network and TCP layers. These results are calculated using the default WEKA threshold of classification, the class with greater than or equal to 0.50 probability is the predicted class. Since precision or other metrics that factor in FPs were not available for direct comparison, there is no way

Table 4.10. Performance Comparison of HTTP-Only Connections

Classifier	All Network		TCP		HTTP		HTTP Extra	
	Recall	UTDR	Recall	UTDR	Recall	UTDR	Recall	UTDR
AdaBoost.BayesNet	0.988366	0.669448	0.990321	0.689026	0.943643	0.557424	0.944072	0.512434
AdaBoost.J48	0.991203	0.609149	0.991847	0.628977	0.962333	0.558093	0.964980	0.558909
BayesNet	0.997258	0.784987	0.997211	0.778210	0.987651	0.608709	0.986793	0.584698
IBk	0.982383	0.518876	0.984433	0.519901	0.947910	0.463429	0.909338	0.903830
J48	0.990965	0.617578	0.991084	0.629060	0.958352	0.548514	0.960236	0.553262
MLP	0.991418	0.551254	0.989010	0.485202	0.879181	0.287256	0.739958	0.256071
SMO	0.931318	0.554768	0.960927	0.569555	0.779913	0.288218	0.904522	0.520780

to conclusively state that classification performance is increased at the lower levels (All Network and/or TCP). The results are interesting nonetheless and pose a possible area for further research, which is out of the scope of this thesis.

4.3 Value Focused Thinking Evaluation Results

This section applies the VFT evaluation approach to the classification results from the All Network dataset using five notional weighting schemes. The process of parsing the PR curve space by alert class is discussed first. Five notional weighting scheme scenarios are presented next, along with the results from the VFT evaluation for each weighting scheme. A summary of the VFT results is then presented to compare classifier selections across the five weighting schemes.

Since the VFT calculations take place after the classifier has classified each instance of the dataset, there is no need to retrain or retest the classifier models when altering a weighting scheme. The VFT evaluation approach selects a threshold setting and provides classifier performance results for the selected threshold using the prediction probabilities provided with the classification results. The results do provide different predictions based on the threshold selected. The point being that weighting schemes can be flexible, adjusted as needed per organizational requirements. For example, different functional offices within the organization could view the same clas-

sification results using a different weighting scheme. Different network sensors could apply different weighting schemes. Weighting schemes could also be used in a manner to tune other classifier parameters to achieve higher performance based on the values of the evaluator rather than tuning to improve overall or marginal performance metrics.

4.3.1 PR Curves by Alert Class.

The evaluation methods used in the previous sections examine PR curves for the overall classification results. The VFT evaluation approach allows an evaluator to weight the importance of detecting malicious connections by alert classifications. In order to weight each alert class and perform the VFT calculations for all thresholds of a PR curve, the single curve was separated into n curves, where n is the number of alert classes in the dataset. Figure 4.5 illustrates the output of this process by showing the normal single PR curve from the classifier performance experiment for AdaBoost.J48 in Figure 4.5(a) compared to the PR curves by alert class in Fig-

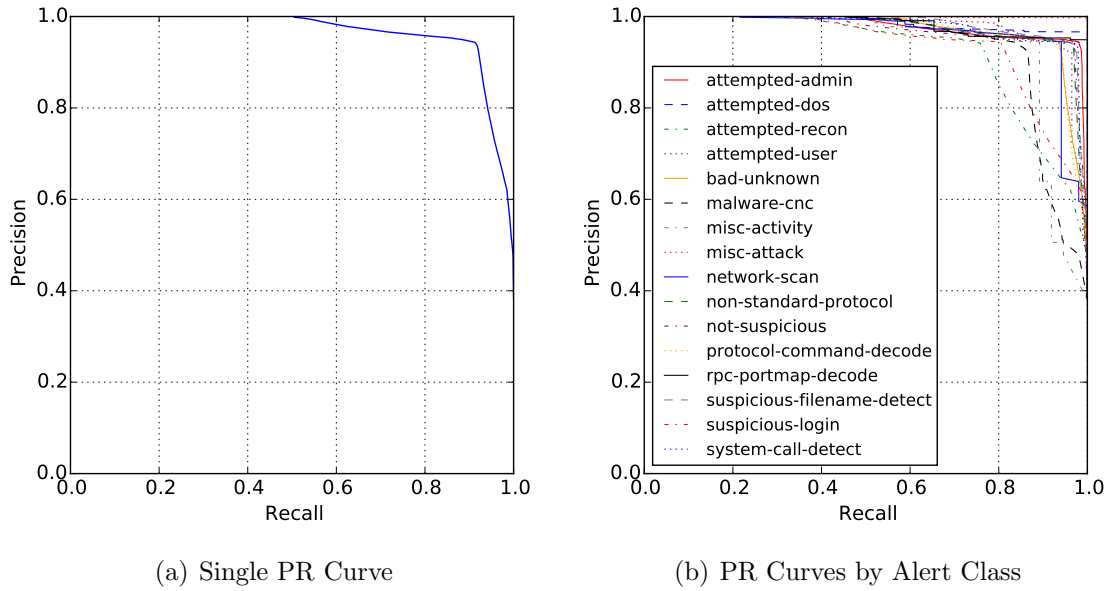


Figure 4.5. PR Curves by Alert Class - Classifier Performance

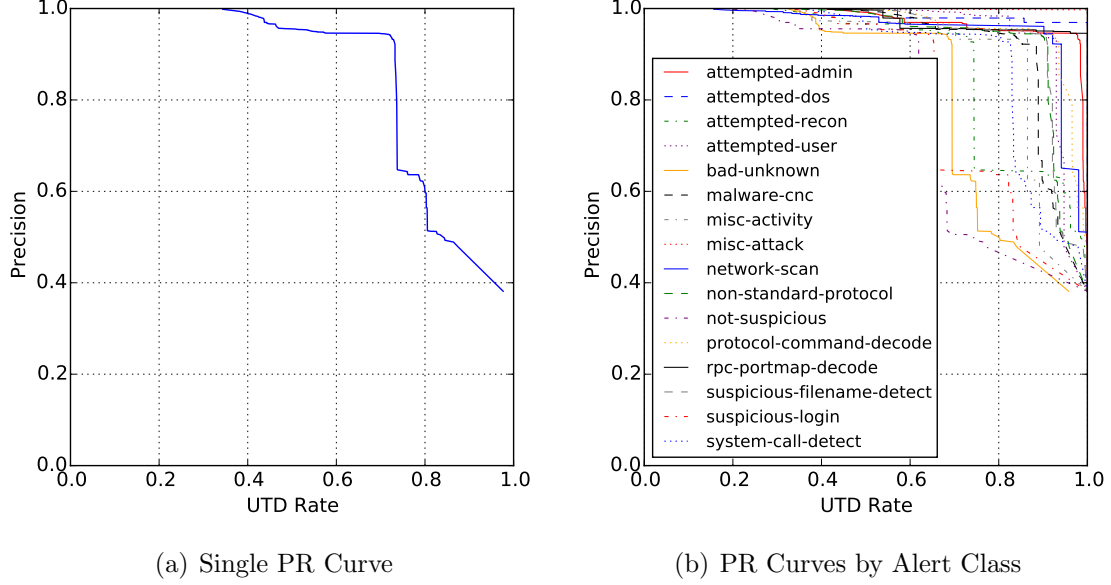


Figure 4.6. PR Curves by Alert Class - Unknown Threat Detection

ure 4.5(b).

A similar method can be used to create PR curves by alert class for the unknown threat detection capability of an algorithm. Because there were no negative class instances (normal connections) used in the testing portion of the unknown threat detection experiments, the precision values calculated at the corresponding thresholds from the classifier performance experiments are used. Figure 4.6 illustrates the output of this process by showing the normal single PR curve incorporating the UTD rate with precision for AdaBoost.J48 in Figure 4.6(a) compared to the PR curves by alert class in Figure 4.6(b).

The data points on the PR curves by alert class provide the detection rates (DR_K and DR_U) and precision rate required to compute the FOM scores for each threshold using the calculation for FOM_{XZ} presented as Equation 3.17. The max scores for each classifier, then the max scores from a set of classifiers can be calculated using Equations 3.18 and 3.19. The weighting schemes, discussed next, become the variables for an evaluator to use when selecting the optimal classifier using the VFT hierarchy.

4.3.2 Scenario 1 - Balanced.

In the balanced scenario, value is placed equally on detection and precision across all of the alert classifications. The specific values for the weighting scheme are presented in Table 4.11. Notice, to achieve balanced weighting on detection and precision, DR_K and DR_U are weighted at 0.25 each, summing to 0.50, and P is weighted at 0.50. To balance the weight across the 16 alert classifications, 0.0625 is used. It is expected that this weighting scheme will select a threshold that represents the PR curve point closest to (1, 1).

Table 4.11. VFT Weighting Scheme - Scenario 1

Tier 1 Weights		Tier 2 Weights	
DR_K	0.25	attempted_admin	0.0625
P	0.50	attempted_dos	0.0625
DR_U	0.25	attempted_recon	0.0625
		attempted_user	0.0625
		bad_unknown	0.0625
		malware_cnc	0.0625
		misc_activity	0.0625
		misc_attack	0.0625
		network_scan	0.0625
		non_standard_protocol	0.0625
		not_suspicious	0.0625
		protocol_command_decode	0.0625
		rpc_portmap_decode	0.0625
		suspicious_filename_detect	0.0625
		suspicious_login	0.0625
		system_call_detect	0.0625

The VFT results for this weighting scheme are provided in Table 4.12, with the highest ranked classifier on the left. The VFT results including the optimal threshold and score are shown, followed by the overall performance rates showing precision, known threat detection (DR_K) and unknown threat detection (DR_U). Then, the detection rates for known and unknown are shown for each alert class for the selected threshold. Similar tables are presented for each weighting scheme scenario.

Table 4.12. VFT Evaluation Results - Scenario 1

		AdaBoost J48		J48		AdaBoost BayesNet		MLP		BayesNet		SMO		IBk	
Value-Focused Results															
Threshold		0.199		0.094		0.292		0.229		0.001		0.241		0.500	
Value-Focused Score		0.6824		0.6813		0.6758		0.6739		0.6725		0.6655		0.5586	
Overall Performance															
P		0.9323		0.9338		0.9200		0.8989		0.8744		0.8914		0.5263	
DR_K		0.9188		0.9164		0.9127		0.9121		0.8928		0.9000		0.9260	
DR_U		0.7292		0.7340		0.7542		0.7299		0.8046		0.7391		0.7341	
DR by Alert Class		DR_K	DR_U	DR_K	DR_U	DR_K	DR_U	DR_K	DR_U	DR_K	DR_U	DR_K	DR_U	DR_K	DR_U
High Pri	attempted-admin	0.9848	0.9818	0.9829	0.9755	0.9773	0.9677	0.9773	0.9725	0.9480	0.8845	0.9703	0.9673	0.9346	0.9473
	attempted-user	0.9474	0.9298	0.9123	0.9123	0.9123	0.8947	0.8947	0.9649	0.9123	0.9123	0.7368	0.7368	0.8421	0.7544
	malware-cnc	0.8559	0.8462	0.8430	0.8387	0.8473	0.8269	0.8720	0.8688	0.8871	0.8871	0.8667	0.8667	0.8581	0.8505
Medium Pri	attempted-dos	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.7143	0.7143
	attempted-recon	0.7634	0.7345	0.7594	0.7313	0.7575	0.7351	0.7527	0.7485	0.7427	0.7368	0.7250	0.7182	0.8786	0.8909
	bad-unknown	0.9387	0.6921	0.9370	0.7008	0.9329	0.7261	0.9360	0.6620	0.9196	0.7953	0.9248	0.6564	0.9384	0.6590
	misc-attack	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	non-std-proto	0.9685	0.9032	0.9668	0.9085	0.9677	0.9091	0.9199	0.8725	0.9188	0.9190	0.9156	0.9170	0.9056	0.7632
	rpc-port-decode	1.0000	1.0000	1.0000	1.0000	1.0000	0.7692	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.8846	0.7308
	susp-file-detect	0.9746	0.9077	0.9701	0.9070	0.9658	0.9530	0.9565	0.9471	0.9575	0.9573	0.9435	0.9453	0.9455	0.8527
	susp-login	0.8108	0.6518	0.8075	0.6474	0.8050	0.6403	0.8089	0.7986	0.7261	0.7141	0.7822	0.7776	0.8830	0.7828
sys-call-detect	0.9780	0.8079	0.9833	0.9744	0.9815	0.8476	0.9912	0.9903	0.9947	0.9947	0.9947	0.9947	0.8819	0.7313	
Low Pri	misc-activity	0.8919	0.8649	0.9189	0.7027	0.9189	0.9189	0.7568	0.7027	0.9459	0.9730	0.7027	0.7027	0.7838	0.7027
	network-scan	0.9412	0.9216	0.9412	0.9020	0.9020	0.9216	0.9608	0.9608	0.9608	0.9608	0.9412	0.9608	0.9804	0.9804
	not-suspicious	0.9714	0.6187	0.9695	0.6164	0.9541	0.6363	0.9616	0.6121	0.8899	0.6971	0.9601	0.7753	0.9296	0.7676
	proto-cmd-decode	0.9349	0.9331	0.9316	0.9243	0.9313	0.9285	0.9334	0.9276	0.9057	0.9061	0.9246	0.9222	0.9173	0.9231

The classifier selection results are similar to the overall and marginal rate comparisons of the All Network dataset from Section 4.2.1, with AdaBoost.J48 and J48 having the highest performance. Recall that in the overall and marginal rate comparisons, WEKA’s default threshold of 0.50 was used to compute the rates. The VFT evaluation shows optimal thresholds, based on the VFT hierarchy and weighting scheme used. Better performance is achieved with lower thresholds than the default of 0.50. For example, the optimal threshold for AdaBoost.J48 is 0.199 with this weighting scheme. This demonstrates that comparing classifier performance at a constant threshold will not always provide the best comparison of performance in relation to the values of the evaluator. The optimal thresholds are marked on the PR curves for the top six classifiers in Figure 4.7.

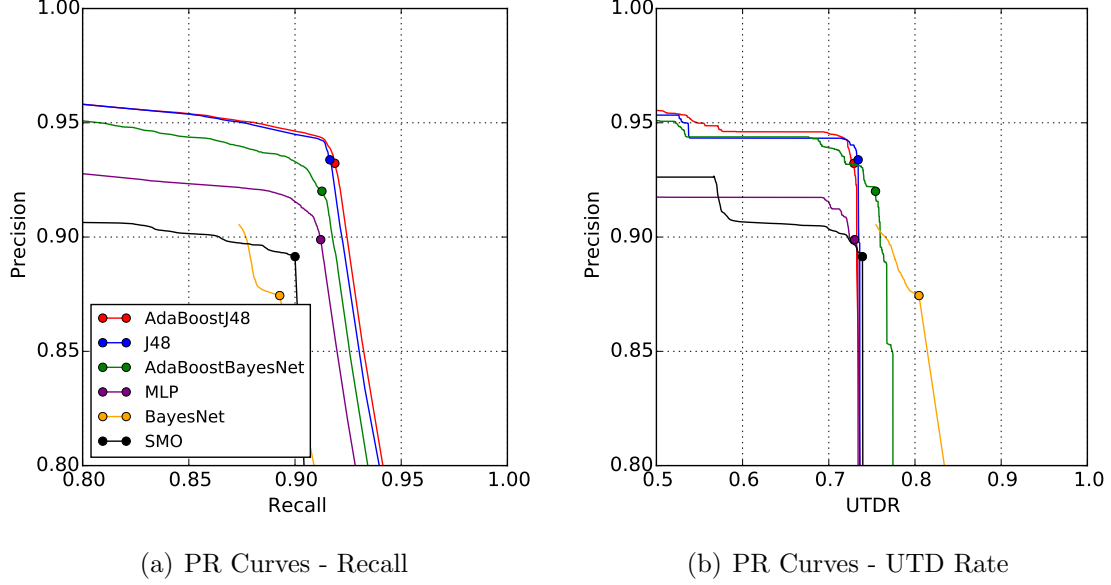


Figure 4.7. VFT Optimal Thresholds - Scenario 1

4.3.3 Scenario 2 - Alert Class Priority.

The alert class priority scenario is a balanced approach with an emphasis placed on the default priorities of the alert classifications. The same weights used for DR_K , DR_U , and P in scenario 1 are applied. Alert classifications are weighted according to their default priority levels provided with the default Snort alert classifications in Appendix A. High priority classes receive a weight of 0.14, medium priority classes receive a weight of 0.06, and low priority classes receive a weight of 0.01. The specific values for the weighting scheme are presented in Table 4.13. This weighting scheme could be considered a first step to take when evaluating PR curves separated by attack class. Using default class priorities to develop the weights removes some of the guess work involved. This weighting scheme still does not take full advantage of weighting known threat detection, unknown threat detection, and precision.

The VFT results for this weighting scheme are provided in Table 4.14. Interestingly, by simply adjusting the alert class weights to differentiate between low, medium, and high priority alerts, the VFT evaluation selects a different optimal threshold for

Table 4.13. VFT Weighting Scheme - Scenario 2

Tier 1 Weights		Tier 2 Weights	
DR_K	0.25	attempted_admin	0.14
P	0.50	attempted_dos	0.06
DR_U	0.25	attempted_recon	0.06
		attempted_user	0.14
		bad_unknown	0.06
		malware_cnc	0.14
		misc_activity	0.01
		misc_attack	0.06
		network_scan	0.01
		non_standard_protocol	0.06
		not_suspicious	0.01
		protocol_command_decode	0.01
		rpc_portmap_decode	0.06
		suspicious_filename_detect	0.06
		suspicious_login	0.06
		system_call_detect	0.06

Table 4.14. VFT Evaluation Results - Scenario 2

		AdaBoost J48		J48		MLP		AdaBoost BayesNet		BayesNet		SMO		IBk	
Value-Focused Results															
Threshold		0.122		0.094		0.229		0.177		0.001		0.241		0.500	
Value-Focused Score		0.6876		0.6864		0.6818		0.6787		0.6724		0.6646		0.5579	
Overall Performance															
P		0.9216		0.9338		0.8989		0.8945		0.8744		0.8914		0.5263	
DR_K		0.9212		0.9164		0.9121		0.9187		0.8928		0.9000		0.9260	
DR_U		0.7323		0.7340		0.7299		0.7619		0.8046		0.7391		0.7341	
DR by Alert Class		DR_K	DR_U	DR_K	DR_U	DR_K	DR_U	DR_K	DR_U	DR_K	DR_U	DR_K	DR_U	DR_K	DR_U
High Pri	attempted-admin	0.9866	0.9844	0.9829	0.9755	0.9773	0.9725	0.9829	0.9759	0.9480	0.8845	0.9703	0.9673	0.9346	0.9473
	attempted-user	0.9649	0.9298	0.9123	0.9123	0.8947	0.9649	0.9298	0.9123	0.9123	0.9123	0.7368	0.7368	0.8421	0.7544
	malware-cnc	0.8656	0.8849	0.8430	0.8387	0.8720	0.8688	0.8602	0.8581	0.8871	0.8871	0.8667	0.8667	0.8581	0.8505
Medium Pri	attempted-dos	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.7143	0.7143
	attempted-recon	0.7685	0.7438	0.7594	0.7313	0.7527	0.7485	0.7653	0.7518	0.7427	0.7368	0.7250	0.7182	0.8786	0.8909
	bad-unknown	0.9409	0.6945	0.9370	0.7008	0.9360	0.6620	0.9377	0.7298	0.9196	0.7953	0.9248	0.6564	0.9384	0.6590
	misc-attack	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	non-std-proto	0.9697	0.9107	0.9668	0.9085	0.9199	0.8725	0.9748	0.9251	0.9188	0.9190	0.9156	0.9170	0.9056	0.7632
	rpc-port-decode	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.8846	1.0000	1.0000	1.0000	1.0000	0.8846	0.7308
	susp-file-detect	0.9760	0.9098	0.9701	0.9070	0.9565	0.9471	0.9784	0.9606	0.9575	0.9573	0.9435	0.9453	0.9455	0.8527
	susp-login	0.8152	0.6539	0.8075	0.6474	0.8089	0.7986	0.8099	0.6599	0.7261	0.7141	0.7822	0.7776	0.8830	0.7828
sys-call-detect	0.9797	0.8264	0.9833	0.9744	0.9912	0.9903	0.9885	0.9057	0.9947	0.9947	0.9947	0.9947	0.8819	0.7313	
Low Pri	misc-activity	0.8919	0.8649	0.9189	0.7027	0.7568	0.7027	0.9189	0.9189	0.9459	0.9730	0.7027	0.7027	0.7838	0.7027
	network-scan	0.9412	0.9412	0.9412	0.9020	0.9608	0.9608	0.9020	0.9216	0.9608	0.9608	0.9412	0.9608	0.9804	0.9804
	not-suspicious	0.9720	0.6194	0.9695	0.6164	0.9616	0.6121	0.9603	0.6430	0.8899	0.6971	0.9601	0.7753	0.9296	0.7676
	proto-cmd-decode	0.9425	0.9358	0.9316	0.9243	0.9334	0.9276	0.9398	0.9377	0.9057	0.9061	0.9246	0.9222	0.9173	0.9231

the highest performer, AdaBoost.J48. The selected threshold was reduced from 0.199 to 0.122, increasing detection rates with a slight reduction in precision, based on the

value placed on detecting alerts in each priority class. By weighting the alert classes in a manner that reflects the values of the evaluator, a threshold is selected that provides better performance with respect to the weighted alert classes. The optimal thresholds are marked on the PR curves for the top six classifiers in Figure 4.8.

A significant discovery with this scenario is that MLP now ranks higher than AdaBoost.BayesNet. A lower prediction threshold of 0.177 was selected for AdaBoost.BayesNet, compared to the threshold of 0.292 selected with the balanced weighting scheme. This improved detection rates, specifically for the high priority classes, but decreased precision causing the overall score to be less than the score of MLP. Even though AdaBoost.BayesNet dominates MLP in PR space in both Figures 4.8(a) and 4.8(b), MLP scores higher in the VFT calculations because of the detection rates per alert class. This demonstrates value in comparing each classifier's performance by alert class using a weighting scheme. Selecting a classifier based solely on overall performance, without parsing by alert classes, could result in the selection of a suboptimal classifier with respect to the alert classes of value.

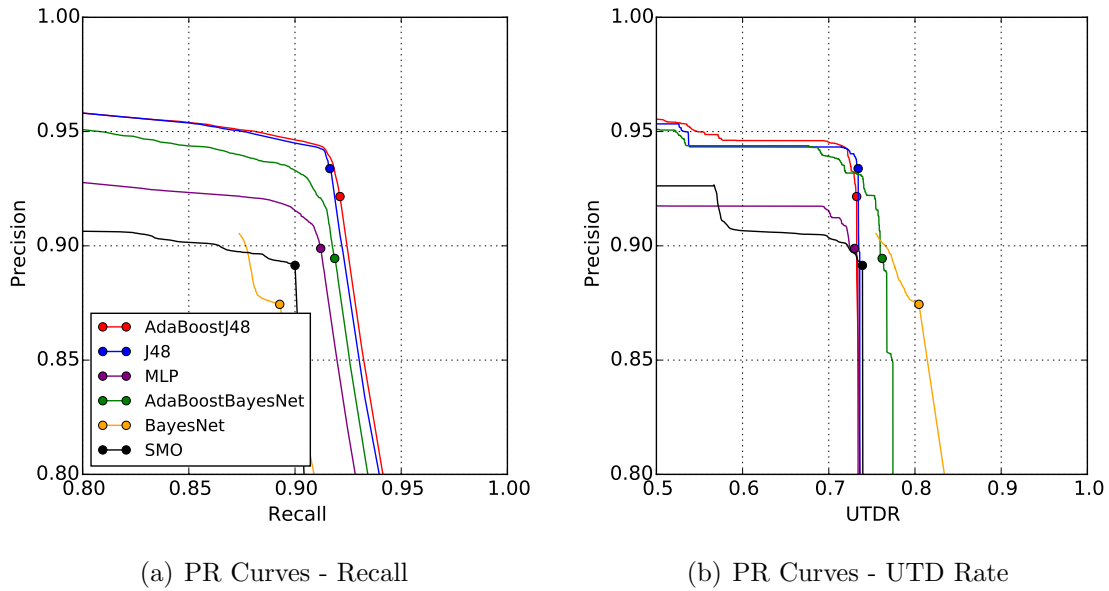


Figure 4.8. VFT Optimal Thresholds - Scenario 2

4.3.4 Scenario 3 - Detect Unknown Threats.

The detect unknown threats scenario uses the same tier 2 weights as scenario 2 to leverage the priority level weighting by alert class, but with tier 1 weights adjusted to emphasize unknown threat detection. DR_K is weighted at 0, while DR_U and P are evenly weighted at 0.50. This weighting scheme could be recommended for a hybrid IDS where the goal is to use the classifier primarily as a means to detect unknown threats, a focus on the set $A_u \cap M_u$ discussed in Section 2.3.4. The complete values for the weighting scheme for scenario 3 are presented in Table 4.15.

The VFT results for this weighting scheme are provided in Table 4.16. MLP ranks the highest with this weighting scheme due to its ability to detect unknown threats of high and medium priority better than the other classifiers. Recall from the marginal rate comparison in Table 4.2 that MLP was noted as a mid-level performer in terms of recall and precision, and as one of the worst performers in terms of UTD rate. Those comparisons were made using the default WEKA threshold of 0.50 and no preference

Table 4.15. VFT Weighting Scheme - Scenario 3

Tier 1 Weights		Tier 2 Weights	
DR_K	0.00	attempted_admin	0.14
P	0.50	attempted_dos	0.06
DR_U	0.50	attempted_recon	0.06
		attempted_user	0.14
		bad_unknown	0.06
		malware_cnc	0.14
		misc_activity	0.01
		misc_attack	0.06
		network_scan	0.01
		non_standard_protocol	0.06
		not_suspicious	0.01
		protocol_command_decode	0.01
		rpc_portmap_decode	0.06
		suspicious_filename_detect	0.06
		suspicious_login	0.06
		system_call_detect	0.06

Table 4.16. VFT Evaluation Results - Scenario 3

		MLP		J48		AdaBoost J48		BayeNet		AdaBoost BayesNet		SMO		IBk	
Value-Focused Results															
Threshold		0.208		0.094		0.122		0.001		0.177		0.241		0.500	
Value-Focused Score		0.6779		0.6761		0.6753		0.6677		0.6675		0.6599		0.5408	
Overall Performance															
P		0.8972		0.9338		0.9216		0.8744		0.8945		0.8914		0.5263	
DR_K		0.9124		0.9164		0.9212		0.8928		0.9187		0.9000		0.9260	
DR_U		0.7315		0.7340		0.7323		0.8046		0.7619		0.7391		0.7341	
DR by Alert Class		DR_K	DR_U	DR_K	DR_U	DR_K	DR_U	DR_K	DR_U	DR_K	DR_U	DR_K	DR_U	DR_K	DR_U
High Pri	attempted-admin	0.9777	0.9766	0.9829	0.9755	0.9866	0.9844	0.9480	0.8845	0.9829	0.9759	0.9703	0.9673	0.9346	0.9473
	attempted-user	0.8947	0.9649	0.9123	0.9123	0.9649	0.9298	0.9123	0.9123	0.9298	0.9123	0.7368	0.7368	0.8421	0.7544
	malware-cnc	0.8731	0.8688	0.8430	0.8387	0.8656	0.8849	0.8871	0.8871	0.8602	0.8581	0.8667	0.8667	0.8581	0.8505
Medium Pri	attempted-dos	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.7143	0.7143
	attempted-recon	0.7533	0.7518	0.7594	0.7313	0.7685	0.7438	0.7427	0.7368	0.7653	0.7518	0.7250	0.7182	0.8786	0.8909
	bad-unknown	0.9363	0.6637	0.9370	0.7008	0.9409	0.6945	0.9196	0.7953	0.9377	0.7298	0.9248	0.6564	0.9384	0.6590
	misc-attack	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	non-std-proto	0.9205	0.8731	0.9668	0.9085	0.9697	0.9107	0.9188	0.9190	0.9748	0.9251	0.9156	0.9170	0.9056	0.7632
	rpc-port-decode	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.8846	1.0000	1.0000	0.8846	0.7308
	susp-file-detect	0.9572	0.9481	0.9701	0.9070	0.9760	0.9098	0.9575	0.9573	0.9784	0.9606	0.9435	0.9453	0.9455	0.8527
	susp-login	0.8090	0.8019	0.8075	0.6474	0.8152	0.6539	0.7261	0.7141	0.8099	0.6599	0.7822	0.7776	0.8830	0.7828
sys-call-detect	0.9921	0.9903	0.9833	0.9744	0.9797	0.8264	0.9947	0.9947	0.9885	0.9057	0.9947	0.9947	0.8819	0.7313	
Low Pri	misc-activity	0.7568	0.7027	0.9189	0.7027	0.8919	0.8649	0.9459	0.9730	0.9189	0.9189	0.7027	0.7027	0.7838	0.7027
	network-scan	0.9608	0.9608	0.9412	0.9020	0.9412	0.9412	0.9608	0.9608	0.9020	0.9216	0.9412	0.9608	0.9804	0.9804
	not-suspicious	0.9620	0.6121	0.9695	0.6164	0.9720	0.6194	0.8899	0.6971	0.9603	0.6430	0.9601	0.7753	0.9296	0.7676
	proto-cmd-decode	0.9334	0.9285	0.9316	0.9243	0.9425	0.9358	0.9057	0.9061	0.9398	0.9377	0.9246	0.9222	0.9173	0.9231

was placed on detecting one alert class over another. The optimal prediction threshold selected for MLP is 0.208, which is lower than the selected thresholds of 0.229 in the previous scenario. Interestingly, the overall DR_U is less than the other classifiers, but the DR_U for the high priority alert classes is collectively higher than the other classifiers allowing MLP to achieve the highest value-focused score. For this example, MLP has better detection capabilities in the alert classes of value, specified by the evaluator. This demonstrates another plausible case where evaluating classifier performance using marginal rates alone, using default thresholds, and not considering detection capabilities by alert class can be ineffective in selecting a classifier for IDS applications. The optimal thresholds are marked on the PR curves for the top six classifiers in Figure 4.9.

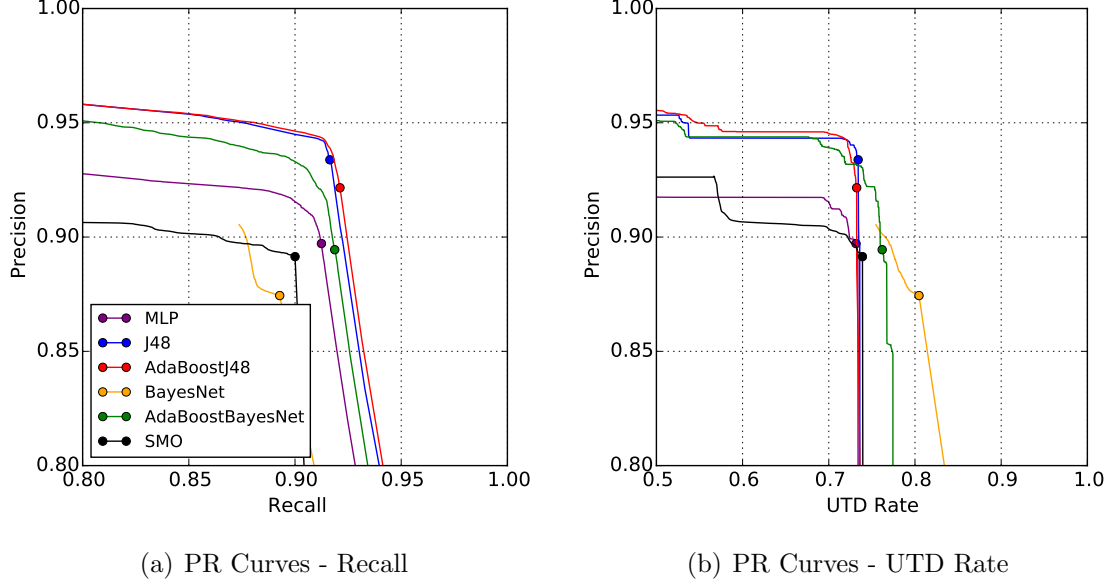


Figure 4.9. VFT Optimal Thresholds - Scenario 3

4.3.5 Scenario 4 - Confirm Known Threats.

The confirm known threats scenario uses the same tier 2 weights as scenarios 2 and 3 to leverage the priority level weighting by alert class, but with tier 1 weights adjusted to emphasize recall from the classifier performance experiments. DR_U is weighted at 0, while DR_K and P are evenly weighted at 0.50. This weighting scheme could be recommended for a hybrid IDS where the goal is to use the classifier primarily as a means to confirm known threats, a focus on the set $A_u \cap M_i$ discussed in Section 2.3.4. The complete values for the weighting scheme for scenario 3 are presented in Table 4.15.

The VFT results for this weighting scheme are provided in Table 4.18 and are very similar to the results from the overall rates from the classifier performance experiments. Although there is still an advantage of using the VFT approach as it considers performance across the range of thresholds selecting the optimal threshold to maximize a balance between precision and known threat detection. This is demonstrated as the thresholds selected for the each classifier differ from the previous scenarios.

Table 4.17. VFT Weighting Scheme - Scenario 4

Tier 1 Weights		Tier 2 Weights	
DR_K	0.50	attempted_admin	0.14
P	0.50	attempted_dos	0.06
DR_U	0.00	attempted_recon	0.06
		attempted_user	0.14
		bad_unknown	0.06
		malware_cnc	0.14
		misc_activity	0.01
		misc_attack	0.06
		network_scan	0.01
		non_standard_protocol	0.06
		not_suspicious	0.01
		protocol_command_decode	0.01
		rpc_portmap_decode	0.06
		suspicious_filename_detect	0.06
		suspicious_login	0.06
		system_call_detect	0.06

Table 4.18. VFT Evaluation Results - Scenario 4

		AdaBoost J48		J48		AdaBoost BayesNet		MLP		BayesNet		SMO		IBk	
Value-Focused Results															
Threshold		0.134		0.139		0.228		0.307		0.001		0.241		0.500	
Value-Focused Score		0.7003		0.6972		0.6929		0.6864		0.6772		0.6692		0.5750	
Overall Performance															
P		0.9242		0.9386		0.9110		0.9026		0.8744		0.8914		0.5263	
DR_K		0.9206		0.9149		0.9158		0.9111		0.8928		0.9000		0.9260	
DR_U		0.7314		0.7312		0.7581		0.7232		0.8046		0.7391		0.7341	
DR by Alert Class		DR_K	DR_U	DR_K	DR_U	DR_K	DR_U	DR_K	DR_U	DR_K	DR_U	DR_K	DR_U	DR_K	DR_U
High Pri	attempted-admin	0.9863	0.9833	0.9829	0.9755	0.9803	0.9755	0.9773	0.9725	0.9480	0.8845	0.9703	0.9673	0.9346	0.9473
	attempted-user	0.9649	0.9298	0.9123	0.9123	0.9298	0.8947	0.8947	0.9298	0.9123	0.9123	0.7368	0.7368	0.8421	0.7544
	malware-cnc	0.8645	0.8527	0.8409	0.8366	0.8559	0.8333	0.8699	0.8634	0.8871	0.8871	0.8667	0.8667	0.8581	0.8505
Medium Pri	attempted-dos	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.7143	1.0000	1.0000	1.0000	1.0000	0.7143	0.7143
	attempted-recon	0.7672	0.7437	0.7586	0.7275	0.7600	0.7361	0.7506	0.7472	0.7427	0.7368	0.7250	0.7182	0.8786	0.8909
	bad-unknown	0.9404	0.6935	0.9365	0.6994	0.9350	0.7295	0.9351	0.6606	0.9196	0.7953	0.9248	0.6564	0.9384	0.6590
	misc-attack	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	non-std-proto	0.9691	0.9093	0.9664	0.9066	0.9703	0.9113	0.9188	0.8718	0.9188	0.9190	0.9156	0.9170	0.9056	0.7632
	rpc-port-decode	1.0000	1.0000	1.0000	1.0000	1.0000	0.7692	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.8846	0.7308
	susp-file-detect	0.9758	0.9092	0.9623	0.9058	0.9751	0.9543	0.9549	0.9009	0.9575	0.9573	0.9435	0.9453	0.9455	0.8527
	susp-login	0.8143	0.6533	0.8068	0.6463	0.8070	0.6554	0.8083	0.7975	0.7261	0.7141	0.7822	0.7776	0.8830	0.7828
sys-call-detect	0.9789	0.8264	0.9815	0.7665	0.9841	0.8493	0.9912	0.9894	0.9947	0.9947	0.9947	0.9947	0.8819	0.7313	
Low Pri	misc-activity	0.8919	0.8649	0.8919	0.7027	0.9189	0.9189	0.7568	0.7027	0.9459	0.9730	0.7027	0.7027	0.7838	0.7027
	network-scan	0.9412	0.9216	0.9412	0.9020	0.9020	0.9216	0.9608	0.9608	0.9608	0.9608	0.9412	0.9608	0.9804	0.9804
	not-suspicious	0.9719	0.6193	0.9692	0.6162	0.9579	0.6399	0.9609	0.6116	0.8899	0.6971	0.9601	0.7753	0.9296	0.7676
	proto-cmd-decode	0.9413	0.9355	0.9313	0.9237	0.9346	0.9322	0.9325	0.9261	0.9057	0.9061	0.9246	0.9222	0.9173	0.9231

For example, valuing known threat detection a threshold of 0.134 is selected for AdaBoost.J48 compared to a threshold of 0.122 that was selected when valuing unknown

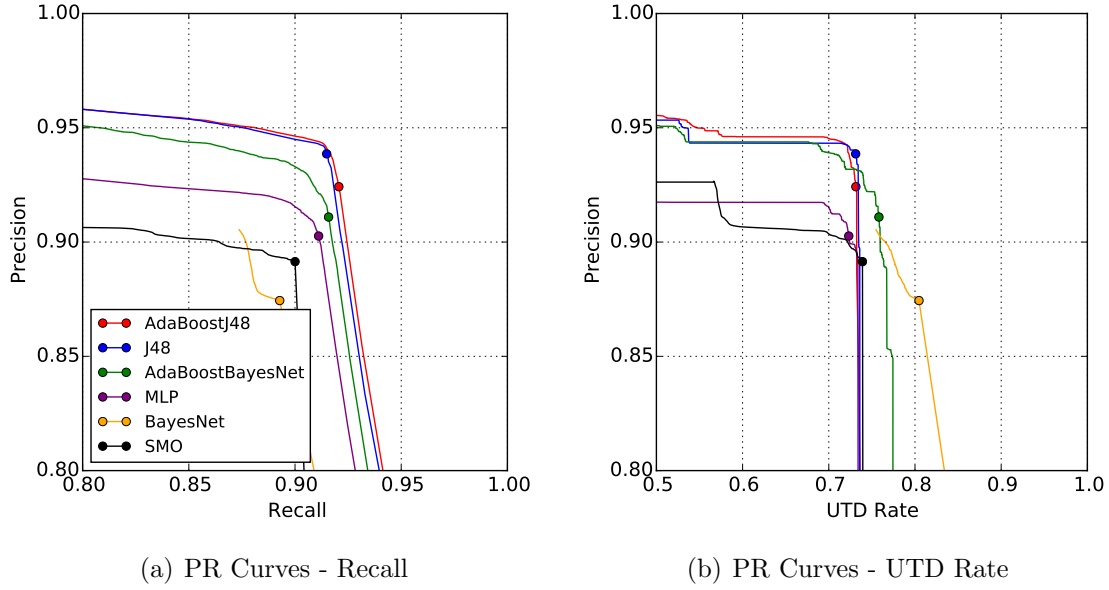


Figure 4.10. VFT Optimal Thresholds - Scenario 4

threat detection. The optimal thresholds are marked on the PR curves for the top six classifiers in Figure 4.10.

4.3.6 Scenario 5 - High Precision.

The high precision scenario uses the same tier 2 weights as scenarios 2, 3, and 4 to leverage the priority level weighting by alert class, but with tier 1 weights adjusted to emphasize precision. DR_U and DR_K are both weighted at 0.10 and P is weighted at 0.80. This weighting scheme could be recommended for use when limited resources are available to review alerts. This weighting scheme should recommend classifiers with thresholds that limit the amount of FPs, but FNs will likely increase causing recall to decrease. The complete values for the weighting scheme for scenario 5 are presented in Table 4.19.

The VFT results for this weighting scheme are provided in Table 4.20. The results show a higher threshold selection resulting in higher precision for six of the seven classifiers when compared to the results from scenario 2. IBk remains at 0.50 because

Table 4.19. VFT Weighting Scheme - Scenario 5

Tier 1 Weights		Tier 2 Weights	
DR_K	0.10	attempted_admin	0.14
P	0.80	attempted_dos	0.06
DR_U	0.10	attempted_recon	0.06
		attempted_user	0.14
		bad_unknown	0.06
		malware_cnc	0.14
		misc_activity	0.01
		misc_attack	0.06
		network_scan	0.01
		non_standard_protocol	0.06
		not_suspicious	0.01
		protocol_command_decode	0.01
		rpc_portmap_decode	0.06
		suspicious_filename_detect	0.06
		suspicious_login	0.06
		system_call_detect	0.06

Table 4.20. VFT Evaluation Results - Scenario 5

		AdaBoost J48		J48		AdaBoost BayesNet		MLP		BayesNet		SMO		IBk	
Value-Focused Results															
Threshold		0.383		0.693		0.558		0.238		0.999		0.389		0.500	
Value-Focused Score		0.7830		0.7824		0.7724		0.7582		0.7543		0.7481		0.5074	
Overall Performance															
P		0.9420		0.9429		0.9349		0.8995		0.9034		0.8947		0.5263	
DR_K		0.9145		0.9111		0.8964		0.9119		0.8752		0.8871		0.9260	
DR_U		0.7218		0.7180		0.7180		0.7244		0.7578		0.7327		0.7341	
DR by Alert Class		DR_K	DR_U	DR_K	DR_U	DR_K	DR_U	DR_K	DR_U	DR_K	DR_U	DR_K	DR_U	DR_K	DR_U
High Pri	attempted-admin	0.9818	0.9781	0.9807	0.9744	0.9592	0.9491	0.9773	0.9725	0.8236	0.8117	0.9562	0.9510	0.9346	0.9473
	attempted-user	0.9474	0.9123	0.9123	0.9123	0.8596	0.8421	0.8947	0.9649	0.9123	0.9123	0.7368	0.7368	0.8421	0.7544
	malware-cnc	0.8441	0.8366	0.8366	0.8333	0.8172	0.8065	0.8710	0.8688	0.8634	0.8624	0.8645	0.8645	0.8581	0.8505
Medium Pri	attempted-dos	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.7143	0.7143
	attempted-recon	0.7578	0.7251	0.7523	0.7211	0.7308	0.7028	0.7521	0.7481	0.7342	0.7287	0.7129	0.7066	0.8786	0.8909
	bad-unknown	0.9365	0.6832	0.9336	0.6802	0.9202	0.6796	0.9358	0.6620	0.8979	0.7506	0.9082	0.6564	0.9384	0.6590
	misc-attack	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	non-std-proto	0.9642	0.8973	0.9603	0.9005	0.9351	0.8576	0.9197	0.8725	0.9148	0.9001	0.9085	0.9034	0.9056	0.7632
	rpc-port-decode	1.0000	1.0000	1.0000	1.0000	1.0000	0.7692	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.8846	0.7308
	susp-file-detect	0.9615	0.9049	0.9586	0.9014	0.9530	0.9398	0.9563	0.9023	0.9552	0.9547	0.9407	0.9379	0.9455	0.8527
	susp-login	0.8060	0.6446	0.8020	0.6414	0.7743	0.6235	0.8087	0.7982	0.7244	0.5803	0.7693	0.7656	0.8830	0.7828
sys-call-detect	0.9692	0.7833	0.9674	0.7577	0.9559	0.8026	0.9912	0.9903	0.9894	0.9885	0.9947	0.9947	0.8819	0.7313	
Low Pri	misc-activity	0.8919	0.7027	0.8649	0.7027	0.8919	0.8919	0.7568	0.7027	0.9189	0.9189	0.7027	0.7027	0.7838	0.7027
	network-scan	0.9412	0.9216	0.9412	0.9020	0.9020	0.9020	0.9608	0.9608	0.9020	0.9020	0.9412	0.9412	0.9804	0.9804
	not-suspicious	0.9700	0.6169	0.9597	0.6094	0.9313	0.6112	0.9615	0.6120	0.7535	0.6092	0.9450	0.7473	0.9296	0.7676
	proto-cmd-decode	0.9292	0.9243	0.9270	0.9064	0.9097	0.9015	0.9331	0.9276	0.9027	0.9024	0.9127	0.9061	0.9173	0.9231

of its non-probabilistic prediction capability with the settings used. The optimal thresholds are marked on the PR curves for the top six classifiers in Figure 4.11. A

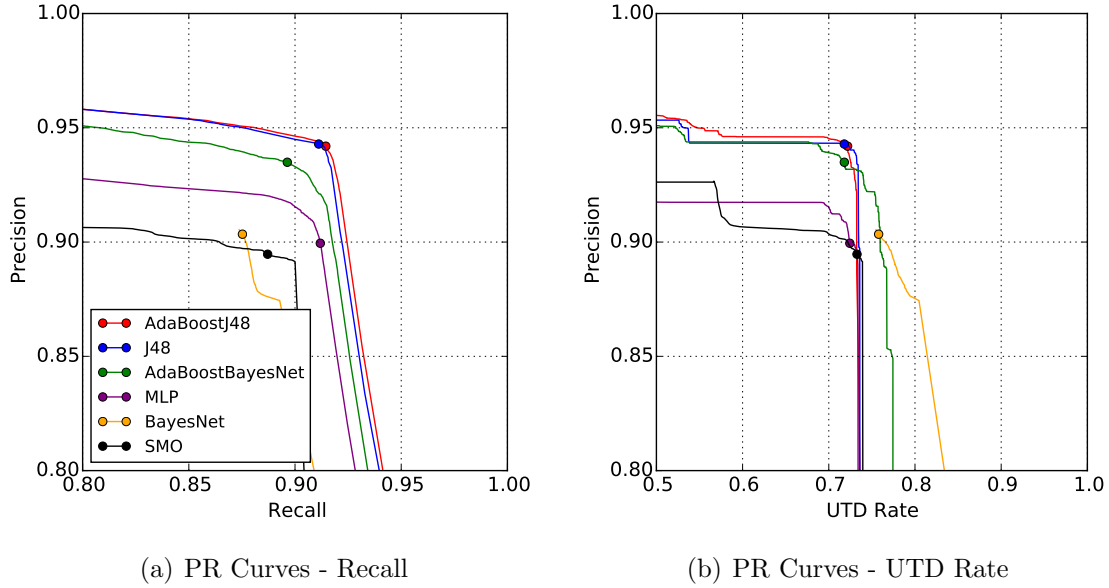


Figure 4.11. VFT Optimal Thresholds - Scenario 5

visual inspection of the curves indicates thresholds selected at the top portion of the curves for 6 of 7 classifiers. MLP is the only classifier where the selected threshold remains at the bottom portion of curve. This is due the VFT calculation maintaining an UTD rate of 0.7244% to optimize the overall score based on the provided weighting scheme.

4.3.7 VFT Results Summary.

A summary of the results from the VFT evaluations is shown in Table 4.21. The results show that classifier selection differed from traditional evaluation methods in 2 of 5 VFT evaluations. With the alert class priority weighting scheme, MLP is selected above AdaBoostBayesNet, and with the detect unknown weighting scheme, MLP is selected above all other classifiers while other classifier rankings shift as well. Results also show that prediction threshold selections varied in 5 of 5 VFT evaluations. These results demonstrate value in applying the VFT evaluation method for classifier selection and prediction threshold selection. The VFT method allows an evaluator

Table 4.21. VFT Weighting Schemes and Results Summary

Weighting Schemes					
	Balanced	Alert Class Priority	Detect Unknown	Detect Known	High Precision
Tier 1 Weights					
DR_K	0.25	0.25	0.00	0.50	0.10
P	0.50	0.50	0.50	0.50	0.80
DR_U	0.25	0.25	0.50	0.00	0.10
Tier 2 Weights					
High Pri	0.0625	0.14	0.14	0.14	0.14
Med Pri	0.0625	0.06	0.06	0.06	0.06
Low Pri	0.0625	0.01	0.01	0.06	0.06
Classifier Selection Results					
1	AdaBoostJ48	AdaBoostJ48	MLP	AdaBoostJ48	AdaBoostJ48
2	J48	J48	J48	J48	J48
3	AdaBoostBayesNet	MLP	AdaBoostJ48	AdaBoostBayesNet	AdaBoostBayesNet
4	MLP	AdaBoostBayesNet	BayesNet	MLP	MLP
5	BayesNet	BayesNet	AdaBoostBayesNet	BayesNet	BayesNet
6	SMO	SMO	SMO	SMO	SMO
7	IBk	IBk	IBk	IBk	IBk

that is unfamiliar with machine learning evaluation techniques to select an optimal classifier and prediction threshold.

4.4 Summary

In summary, this chapter presented the results and analysis for experiments conducted per the methodology discussed in Chapter III. First, traditional evaluation methods of comparing overall rates, marginal rates, and PR curves were explored on four subsets of the dataset. Next, the proposed VFT evaluation method was applied to one of the four dataset subsets, the All Network dataset, using five notional weighting schemes. A summary of these results is presented below.

AdaBoost.J48 and J48 consistently outperformed the other algorithms when comparing the overall rates of F-Measure, G-Measure, and MCC, with the highest scores for these mean overall rates achieved using the HTTP dataset with additional features. J48 achieved an F-Measure of 0.9705 with a 95% CI between 0.9692 and 0.9719, a G-Measure of 0.9706 with a 95% CI between 0.9692 and 0.9720, and an MCC of 0.9623 with a 95% CI between 0.9606 and 0.9641. AdaBoost.J48 achieved

an F-Measure of 0.9691 with a 95% CI between 0.9676 and 0.9706, a G-Measure of 0.9691 with a 95% CI between 0.9676 and 0.9706, and an MCC of 0.9603 with a 95% CI between 0.9584 and 0.9622.

Comparisons made to determine if classifier performance was better at lower or higher layers of the TCP/IP stack were inconclusive. The inconsistency of the negative class (normal connections) across the four datasets caused the inability to compare performance metrics that incorporated the negative class metrics in the calculations. Comparisons of recall and UTD rates were accomplished and showed no significant value in generalization between the All Network and TCP datasets when comparing only TCP connections. However, there were significant improvements noted in the recall and UTD rates in the All Network and TCP datasets over the HTTP datasets when comparing only HTTP connections. This significance, while promising, remains inconclusive without the ability to compare measures relating to FPs.

Incorporating the detection rates from the unknown threat detection experiment to compare with rates of precision and recall resulted in a less conclusive classifier selection decision. While AdaBoost.J48 and J48 performed well overall in terms of precision and recall, mid to low performance was noted in their ability to detect unknown threats when compared to the other classifiers across the four datasets. BayesNet consistently performed the best at detecting unknown threats, achieving the highest UTD rate of 0.7808 with a 95% CI between 0.7788 and 0.7829 on the TCP dataset. The difficulty in selecting a classifier using precision, recall, and UTD rates demonstrated potential value in allowing a evaluator to weight their own value on these metrics during the evaluation process. Also noted was the need to evaluate across the range of prediction thresholds, rather than a default constant threshold for each classifier, and to evaluate detection capability by alert classification, as some

alerts exhibit higher potential damage than others.

The VFT evaluation results demonstrated an engineering advantage in evaluating classifiers using five notional weighting schemes. The balanced weighting scheme, scenario 1, returned results similar to the overall rate comparisons as expected. Using a weight scheme that incorporated alert priorities, scenario 2, recommended a different optimal threshold for the highest scoring classifier, AdaBoost.J48. This scenario also demonstrated the value of parsing a PR curve into multiple curves representing different alert classes as it ranked a MLP above AdaBoost.BayesNet even though AdaBoost.BayesNet dominated the traditional PR space using a single curve approach. This evaluation fallacy has not been demonstrated prior to this research and is a significant finding that should be considered when evaluating classifiers for use in intrusion detection. The remaining scenarios continued to demonstrate value added by applying other plausible weighting schemes focused on detecting unknown threats, confirming known threats, and high precision, can lead to the selection of different classifiers with different optimal thresholds. Results are summarized by concluding that classifier selection differed from traditional evaluation methods in 2 of 5 VFT evaluations and prediction threshold selections varied in 5 of 5 VFT evaluations.

V. Conclusion

5.1 Chapter Overview

This chapter concludes this thesis by summarizing the research goals, followed by the contributions made through this research. Finally, recommendations for potential future research are presented.

5.2 Conclusions of Research

The primary goal of this research was to provide a classifier evaluation methodology which incorporates a weighted scheme to allow an evaluator to weight metrics of value to use when evaluating classifiers for use in hybrid IDS architectures. A Value-Focused Thinking (VFT) methodology was applied with a value hierarchy created specifically for the use of machine learning classifiers in hybrid IDS configurations. This enabled an evaluator to weight precision, recall, and unknown threat detection (UTD) rates to aid in the selection of a classifier and prediction threshold. Recall and UTD rates were further divided by alert classification, allowing an evaluator to weight the importance of detecting malicious connections by alert class.

The second goal of this research was to demonstrate the effectiveness of the VFT approach, five notional weighting schemes were used in the VFT evaluation and the results were compared to typical methods of classifier evaluation. The analysis of these results demonstrated an engineering advantage to the use of the VFT approach when evaluating threat classifiers as it selects classifiers and thresholds optimized for performance that best represents the values of the evaluator.

The final goal of this research was to compare classifier performance at various layers in the TCP/IP stack to determine if threats generalize better or worse based on the level of data used to train the classifier. While classification performance

related to the positive class was better at the lower layers than the application layers for HTTP threats, inconsistencies in the negative classes of the test sets provided inconclusive results overall.

5.3 Research Contributions

There are three main research contributions to the application of machine learning to the field of intrusion detection. First is the comprehensive approach, similar to cross-validation, used to evaluate a classifier’s ability to detect unknown threats. The comprehensive nature of this method reduces the likelihood of misrepresentation by allowing every threat the opportunity to be tested as an unknown threat. The outcome of this is a more realistic metric to represent a classifier’s ability to detect unknown threats.

The second contribution is the method used to separate precision-recall (PR) curves by alert class for classifier performance evaluation. Alerts have different priority levels, by not considering this when evaluating PR curves, or ROC curves for that matter, researchers and vendors may misrepresent the desired functionality of their classifiers. Evaluators should question the capability of a classifier to detect specific types or classes of attacks when the system has been trained on them (known threats) and when they have not been trained on them (unknown threats).

Finally, the third contribution is the application of the VFT methodology to incorporate the two previous contributions into a calculation that can be used by an evaluator to select a classifier and an optimal threshold setting to use in a hybrid IDS configuration.

5.4 Recommendations for Future Research

There are three recommendations for future research to expand on this research. The first recommendation is to extend methodology to compare the use of multiple classifiers and decision-level fusion methods. While this research was limited to selecting only one classifier and optimal threshold, combining predictions from multiple classifiers has been shown to increase overall performance [?]. Predictions can be fused in a naïve manner using logical AND or OR conditions or by using more complex fusing methods. The VFT method can test all classifier combinations at all threshold levels to find the optimum combination of classifiers and thresholds best suited to the provided weighting scheme.

A second recommendation is to use the evaluation methods from this research to tune classifier settings. This research utilized mostly default classifier configurations. Typically, evaluation metrics such as accuracy, precision, or recall are used to optimize the classifier configurations. The VFT score and/or UTD rates used in this research could be used as evaluation metrics to tune classifiers to maximum performance specific to the value hierarchy and weighting scheme of the evaluator rather than optimizing to traditional evaluation metrics. The result would be classifier optimized to the evaluator’s values versus a one-size-fits-all approach.

The final recommendation for future research is to incorporate more meaning into the alert classification of the Snort rule set. This research considered only the default Snort alert classes. Creating a taxonomy of alert classifications based with the intent of weighting them as values in a VFT evaluation methodology would yield more meaningful PR curves for each alert class. Theoretically, this would better align the evaluation method to meaningful values of the evaluator thus providing more useful results than would be provided with the default alert classes.

Appendix A. Default Snort Alert Classifications

Table A.1. Default Snort Alert Classifications [6]

Classtype	Description	Priority
attempted-admin	Attempted Administrator Privilege Gain	high
attempted-user	Attempted User Privilege Gain	high
inappropriate-content	Inappropriate Content was Detected	high
policy-violation	Potential Corporate Privacy Violation	high
shellcode-detect	Executable code was detected	high
successful-admin	Successful Administrator Privilege Gain	high
successful-user	Successful User Privilege Gain	high
trojan-activity	A Network Trojan was detected	high
unsuccessful-user	Unsuccessful User Privilege Gain	high
web-application-attack	Web Application Attack	high
attempted-dos	Attempted Denial of Service	medium
attempted-recon	Attempted Information Leak	medium
bad-unknown	Potentially Bad Traffic	medium
default-login-attempt	Attempt to login by a default username and password	medium
denial-of-service	Detection of a Denial of Service Attack	medium
misc-attack	Misc Attack	medium
non-standard-protocol	Detection of a non-standard protocol or event	medium
rpc-portmap-decode	Decode of an RPC Query	medium
successful-dos	Denial of Service	medium
successful-recon-largescale	Large Scale Information Leak	medium
successful-recon-limited	Information Leak	medium
suspicious-filename-detect	A suspicious filename was detected	medium
suspicious-login	An attempted login using a suspicious username was detected	medium
system-call-detect	A system call was detected	medium
unusual-client-port-connection	A client was using an unusual port	medium
web-application-activity	Access to a potentially vulnerable web application	medium
Continued on next page		

Classtype	Description	Priority
icmp-event	Generic ICMP event	low
misc-activity	Misc activity	low
network-scan	Detection of a Network Scan	low
not-suspicious	Not Suspicious Traffic	low
protocol-command-decode	Generic Protocol Command Decode	low
string-detect	A suspicious string was detected	low
unknown	Unknown Traffic	low
tcp-connection	A TCP connection was detected	very low

Appendix B. Network Sensor Configuration

This appendix explains the configurations of Security Onion, Bro, and Snort used in this research.

Security Onion Setup

Using VMWare Workstation (version 10.0.1), a virtual machine was created of Security Onion (version 12.04.4). The hardware settings for the virtual machine shown in the Table B.1. Updates to the Security Onion system were completed using the soup update utility provided by the Security Onion development team. The specific versions of applications used in this research are shown in Table B.2.

There are two setup phases for Security Onion which are accomplished using the preloaded setup configuration found on the desktop. During the first phase of setup, the network interfaces are configured so Eth0 is the management interface and Eth1 is the sniffing interface. These interfaces correspond to the VMWare network adapters 1 and 2, respectively. After rebooting, the second phase of setup is completed using the setup configuration found on the desktop again, bypassing the network configuration portion this time. The advanced setup menu option is used to create a standalone configuration. Snort is selected as the IDS engine using the “VRT and ET No/GPL” rule set. Bro and ELSA are enabled in this configuration as well. Other applications such as http_agent, Argus, Prads, and Salt are not enabled as they are not used in

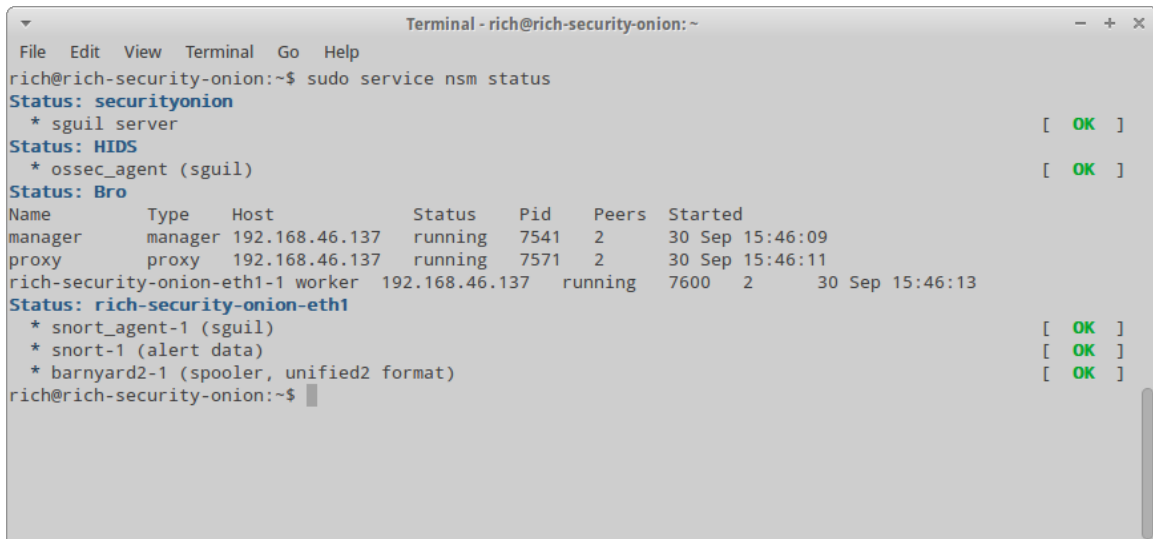
Table B.1. Security Onion Virtual Machine Hardware Settings

Hardware Component	Specification
Memory	4 GB
Processors	1
Hard Disk	100 GB
Network Adapter 1	NAT to Host
Network Adapter 2	VMNet2

Table B.2. Security Onion Applications

Application	Version
Bro	2.3.1
ELSA	Sphinx 2.0.7-id64-dev
PulledPork	0.7.0
Python	2.7.3
Sguil	0.8.0
Snorby	2.6.2
Snort	2.9.6.2
Tcpreplay	3.4.3

this research. The reader can learn more about each of these applications in [39]. After setup is complete, the status of the configured services can be seen using the `sudo service nsm status` command as shown in Figure B.1.



```
Terminal - rich@rich-security-onion: ~
File Edit View Terminal Go Help
rich@rich-security-onion:~$ sudo service nsm status
Status: securityonion
* sgul server [ OK ]
Status: HIDS
* ossec_agent (sguil) [ OK ]
Status: Bro
Name      Type      Host           Status    Pid    Peers  Started
manager   manager   192.168.46.137 running   7541    2      30 Sep 15:46:09
proxy     proxy     192.168.46.137 running   7571    2      30 Sep 15:46:11
rich-security-onion-eth1-1 worker 192.168.46.137 running   7600    2      30 Sep 15:46:13
Status: rich-security-onion-eth1
* snort_agent-1 (sguil) [ OK ]
* snort-1 (alert data) [ OK ]
* barnyard2-1 (spooler, unified2 format) [ OK ]
rich@rich-security-onion:~$
```

Figure B.1. Checking Status of Security Onion Services

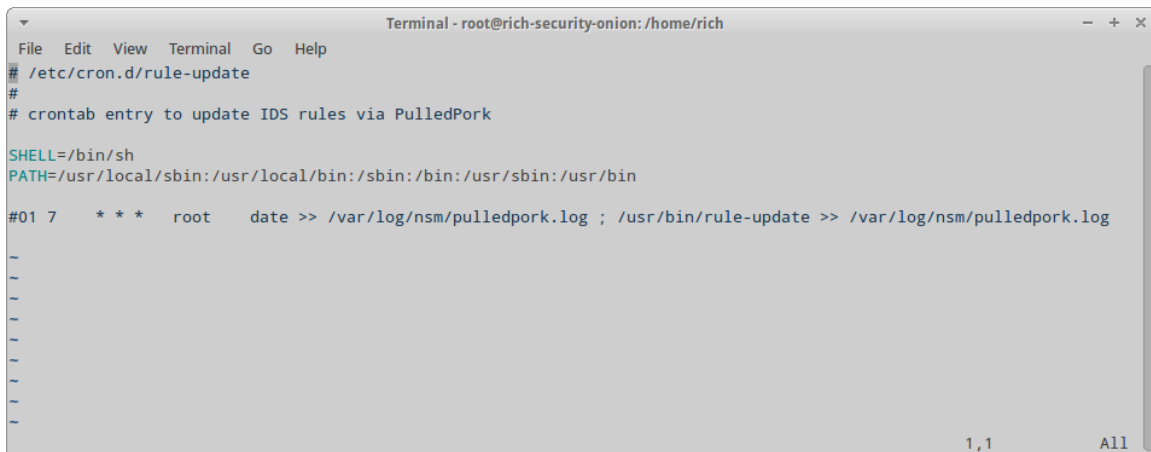
Snort Configuration

Snort was configured as the IDS and the rule set was selected in the Security Onion setup. Additional steps include verifying the rule set, disabling rule updates

(a control for the experiments), and modifying the alert output to the Unified2 file format.

The downloaded Snort rule set is verified using the command `grep -v '^#' /etc/nsm/rules/downloaded.rules | sed '/^$/d' > output.rules` which deletes rules that are commented out and empty lines. The total rule count for the VRT and ET No/GPL rule set used in this research is 21,641.

By default, PulledPork performs a daily rule update using cron job with the script `/usr/bin/rule-update`. While useful for an operational network sensor, to maintain consistency throughout the experiments, no rule updates were allowed. The cron job was disabled by commenting out the crontab entry in `/etc/cron.d/rule-update` as shown in Figure B.2.



```
Terminal - root@rich-security-onion: /home/rich
File Edit View Terminal Go Help
# /etc/cron.d/rule-update
#
# crontab entry to update IDS rules via PulledPork
SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin
#01 7 * * * root date >> /var/log/nsm/pulledpork.log ; /usr/bin/rule-update >> /var/log/nsm/pulledpork.log
~
~
~
~
~
~
~
1,1 All
```

Figure B.2. Disabling Daily Rule Update Script

Output to the Unified2 format is enabled by default in Security Onion and can be verified in the `snort.conf`. The default directory where the Unified2 alert logs are written, which is needed for the Bro configuration portion, is `/nsm/sensor_data/sensor_name/snort-1/`.

Bro Configuration

Bro was configured to output additional connection-level features shown in Table 3.3. To log the additional features, a copy of the script used to create the `conn.log` (`/opt/bro/share/bro/base/protocols/conn/main.bro`) was modified to produce another log file containing the additional features. The modified script is then saved in the local site directory (`/opt/bro/share/bro/site`) and the `local.bro` script, in the same directory, is updated to load the script with the `@load` operation. Updating the `local.bro` script is required to call any additional scripts beyond the default scripts.

One field in particular, `port_service`, was added using a script made available on the Bro developer's GitHub site [22]. Readers experimenting with Bro scripting should refer to the developer's and other community member's GitHub sites for example code. By default, Bro uses dynamic protocol analyzers to identify the service of each connection. During the data processing stage, it was noticed that many of the connections had a missing value for the service field. Using this script to add the `port_service` field provided the ability of identifying the service by port number in the cases where the service field was blank.

Unified2 logging was also enabled in Bro, allowing Snort alert logs to be in the same format as the connection-level logs produced by Bro. Enabling Unified2 logging is a two-step process. The first step is to enable the included Unified2 logging capability by adding the script `/base/files/unified2/main.bro` to the `local.bro` script using the `@load` operation. The second step is to create an additional script to redefine the directory Bro should watch for Unified2 alert logs and the directories where the Snort signature map, generator map, and alert classification configurations are stored.

Appendix C. WEKA Commands and Options

This appendix includes the options called for each WEKA function call used in this research. Brief explanations of options are included, but full details for each option and unused options can be found in the WEKA API documentation [8] and the companion book [46].

Classifier Configuration Options:

BayesNet.

BayesNet is used without an all-dimensions tree data structure, an optional learning option that can speed up the learning process. The K2 search algorithm is used with the default of 1 used for maximum number of parents and Bayes as the score type. The default SimpleEstimator is used to estimate the conditional probability tables of the Bayes network directly from the training data. The exact WEKA command used with options is below:

```
weka.classifiers.bayes.BayesNet -- -D \  
-Q weka.classifiers.bayes.net.search.local.K2 -- -P 1 -S BAYES \  
-E weka.classifiers.bayes.net.estimate.SimpleEstimator -- -A 0.5
```

AdaBoostM1.BayesNet.

AdaBoostM1.BayesNet is used with a 100% weight mass for training. This percentage can be decreased to speed up the learning process. AdaBoost runs for 10 iterations. The same BayesNet classifier settings previously discussed are used with the AdaBoost algorithm. The exact WEKA command used with options is below:

```
weka.classifiers.meta.AdaBoostM1 -- -P 100 -S 1 -I 10 \  
-W weka.classifiers.bayes.BayesNet -- -D \  

```

```
-Q weka.classifiers.bayes.net.search.local.K2 -- -P 1 -S BAYES \  
-E weka.classifiers.bayes.net.estimate.SimpleEstimator -- -A 0.5
```

IB k .

IB k is used with a k of 1, a brute force linear nearest neighbor search, and a standard Euclidean function to measure distance. The exact WEKA command used with options is below:

```
weka.classifiers.lazy.IBk -- -K 1 -W 0 \  
-A "weka.core.neighboursearch.LinearNNSearch \  
-A "\"weka.core.EuclideanDistance -R first-last\""
```

SMO.

SMO is used with the default settings for complexity constant, tolerance parameter, and epsilon round-off. The -M option is added to fit logistic models to the SVM outputs, enabling a probability estimate to be included with each prediction. The default SVM kernel, PolyKernel, is used with the default options. The exact WEKA command used with options is below:

```
weka.classifiers.functions.SMO -- -C 1.0 -L 0.001 -P 1.0E-12 \  
-N 0 -M -V -1 -W 1 \  
-K "weka.classifiers.functions.supportVector.PolyKernel \  
-C 250007 -E 1.0"
```

J48.

J48 is used with the default confidence threshold for pruning of 0.25 and a minimum of 2 instances per leaf. The exact WEKA command used with options is below:

```
weka.classifiers.trees.J48 -- -C 0.25 -M 2
```

AdaBoostM1.J48.

AdaBoostM1.J48 is used with a 100% weight mass for training and runs for 10 iterations. The same J48 classifier settings previously discussed are used with the AdaBoost algorithm. The exact WEKA command used with options is below:

```
weka.classifiers.meta.AdaBoostM1 -- -P 100 -S 1 -I 10 \  
-W weka.classifiers.trees.J48 -- -C 0.25 -M 2
```

MLP.

MLP with backpropagation is used with the default settings for learning rate and momentum rate. The default number of epochs, 500, is also used. The exact WEKA command used with options is below:

```
weka.classifiers.functions.MultilayerPerceptron -- \  
-L 0.3 -M 0.2 -N 500 -V 0 -S 0 -E 20 -H a
```

Data Preprocessing Commands and Options:

Random Sampling.

Random sampling is performed on the All Network and TCP datasets using the command and options:

```
weka.filters.supervised.instance.Resample -i <dataset.arff> \  
-o <dataset-10%.arff> -c last -Z 10 -no-replacement -B 1
```

Cross-Validation Training and Test Sets.

Training sets for the stratified cross-validation folds are created using the command and options:

```
weka.filters.supervised.instance.StratifiedRemoveFolds \  
  -i <input-file> -o <output-training-file> -c last -N 10 \  
  -F <fold-number> -V
```

Test sets for the stratified cross-validation folds are created using the command and options:

```
weka.filters.supervised.instance.StratifiedRemoveFolds \  
  -i <input-file> -o <output-test-file> -c last -N 10 \  
  -F <fold-number>
```

Appendix D. Confusion Matrices

The confusion matrices in this appendix represent the total number of instances classified across the 10 folds of the cross validation experiments for each dataset and classifier.

Table D.1. Confusion Matrix - All Network Data

Classifier	True Positives	False Positives	True Negatives	False Negatives
AdaBoost.BayesNet	132,276	9,734	227,864	14,255
AdaBoost.J48	133,810	8,063	229,535	12,721
BayesNet	128,810	15,818	221,780	17,721
IBk	135,689	122,146	115,452	10,842
J48	133,767	8,193	229,405	12,764
MLP	133,036	13,227	224,371	13,495
SMO	128,309	14,714	222,884	18,222

Table D.2. Confusion Matrix - TCP Data

Classifier	True Positives	False Positives	True Negatives	False Negatives
AdaBoost.BayesNet	132,070	8,941	189,024	13,174
AdaBoost.J48	132,948	7,295	190,670	12,296
BayesNet	128,528	14,222	183,743	16,716
IBk	136,305	119,374	78,591	8,939
J48	132,841	7,392	190,573	12,403
MLP	132,294	11,424	186,541	12,950
SMO	129,770	13,441	184,524	15,474

Table D.3. Confusion Matrix - HTTP Data

Classifier	True Positives	False Positives	True Negatives	False Negatives
AdaBoost.BayesNet	39,583	1,619	144,805	2,364
AdaBoost.J48	40,367	1,231	145,193	1,580
BayesNet	41,428	13,259	133,165	519
IBk	39,762	2,226	144,198	2,185
J48	40,200	901	145,523	1,747
MLP	36,879	2,532	143,892	5,068
SMO	32,693	7,288	139,136	9,254

Table D.4. Confusion Matrix - HTTP Data with Additional Features

Classifier	True Positives	False Positives	True Negatives	False Negatives
AdaBoost.BayesNet	39,601	1,510	144,914	2,346
AdaBoost.J48	40,478	1,113	145,311	1,469
BayesNet	41,393	13,062	133,362	554
IBk	38,144	66,244	80,180	3,803
J48	40,279	777	145,647	1,668
MLP	31,039	2,985	143,439	10,908
SMO	37,942	3,196	143,228	4,005

Table D.5. Confusion Matrix - FTP Data

Classifier	True Positives	False Positives	True Negatives	False Negatives
AdaBoost.BayesNet	42,817	75,127	34,280	5,999
AdaBoost.J48	1,177	683	108,724	47,639
BayesNet	42,817	75,127	34,280	5,999
IBk	17,568	32,177	77,230	31,248
J48	704	136	109,271	48,112
MLP	552	234	109,173	48,264
SMO	110	40	109,367	48,706

Table D.6. Confusion Matrix - FTP Data with Additional Features

Classifier	True Positives	False Positives	True Negatives	False Negatives
AdaBoost.BayesNet	42,817	75,127	34,280	5,999
AdaBoost.J48	1,133	611	108,796	47,683
BayesNet	42,817	75,127	34,280	5,999
IBk	48,704	98,418	10,989	112
J48	703	137	109,270	48,113
MLP	647	229	109,178	48,169
SMO	144	40	109,367	48,672

Bibliography

- [1] “Bro.” [Online]. Available: <https://www.bro.org/>
- [2] “Bro documentation.” [Online]. Available: <https://www.bro.org/documentation/index.html>
- [3] “Emerging threats.” [Online]. Available: <http://emergingthreats.net/>
- [4] “Intrusion [def. 1],” Merriam-Webster Online. [Online]. Available: <http://www.merriam-webster.com/dictionary/intrusion>
- [5] “Snort.” [Online]. Available: <https://www.snort.org/>
- [6] “Snort user’s manual.” [Online]. Available: <http://manual.snort.org/>
- [7] “Talos.” [Online]. Available: <https://www.snort.org/talos>
- [8] “WEKA API Documentation.” [Online]. Available: <http://weka.sourceforge.net/doc.dev/>
- [9] D. W. Aha, D. Kibler, and M. K. Albert, “Instance-based learning algorithms,” *Machine learning*, vol. 6, no. 1, pp. 37–66, 1991.
- [10] D. Anderson, T. Frivold, and A. Valdes, *Next-generation intrusion detection expert system (NIDES): A summary*. SRI International, Computer Science Laboratory, 1995.
- [11] J. P. Anderson, “Computer security threat monitoring and surveillance,” Technical report, James P. Anderson Company, Fort Washington, Pennsylvania, Tech. Rep., 1980.
- [12] J. M. Beaver, C. T. Symons, and R. E. Gillen, “A learning system for discriminating variants of malicious network traffic,” in *Proceedings of the Eighth Annual Cyber Security and Information Intelligence Research Workshop*. ACM, 2013, p. 23.
- [13] C. Cortes and V. Vapnik, “Support-vector networks,” *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.
- [14] J. Davis and M. Goadrich, “The relationship between Precision-Recall and ROC curves,” in *Proceedings of the 23rd international conference on Machine learning*. ACM, 2006, pp. 233–240.
- [15] D. E. Denning, “An intrusion-detection model,” *Software Engineering, IEEE Transactions on*, no. 2, pp. 222–232, 1987.

- [16] S. Dua and X. Du, *Data Mining and Machine Learning in Cybersecurity*. CRC Press, 2014.
- [17] C. Elkan, “Results of the KDD’99 classifier learning,” *ACM SIGKDD Explorations Newsletter*, vol. 1, no. 2, pp. 63–64, 2000.
- [18] R. Fielding and J. Reschke, “IETF RFC 7231 Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content,” 2014.
- [19] Y. Freund, R. E. Schapire *et al.*, “Experiments with a new boosting algorithm,” in *International Conference on Machine Learning*, vol. 96, 1996, pp. 148–156.
- [20] L. M. Garcia, “Programming with libpcap \pm sniffing the network from our own application,” *Hakin9-Computer Security Magazine*, pp. 2–2008, 2008.
- [21] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, “The WEKA data mining software: an update,” *ACM SIGKDD explorations newsletter*, vol. 11, no. 1, pp. 10–18, 2009.
- [22] S. Hall, “bro-scripts,” GitHub. [Online]. Available: <https://github.com/sethhall/bro-scripts>
- [23] H. He and E. A. Garcia, “Learning from imbalanced data,” *Knowledge and Data Engineering, IEEE Transactions on*, vol. 21, no. 9, pp. 1263–1284, 2009.
- [24] R. L. Keeney, “Value-focused thinking: Identifying decision opportunities and creating alternatives,” *European Journal of operational research*, vol. 92, no. 3, pp. 537–549, 1996.
- [25] M. V. Mahoney and P. K. Chan, “An analysis of the 1999 DARPA/Lincoln Laboratory evaluation data for network anomaly detection,” in *Recent Advances in Intrusion Detection*. Springer, 2003, pp. 220–237.
- [26] J. McHugh, “Testing intrusion detection systems: a critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln Laboratory,” *ACM transactions on Information and system Security*, vol. 3, no. 4, pp. 262–294, 2000.
- [27] R. F. Mills, M. R. Grimaila, G. L. Peterson, and J. W. Butts, “A scenario-based approach to mitigating the insider threat,” *Information Systems Security Association*, vol. 9, no. 4, pp. 12–19, May 2011.
- [28] MIT Lincoln Laboratory, “DARPA Intrusion Detection Data Sets.” [Online]. Available: <http://www.ll.mit.edu/mission/communications/cyber/CSTcorpora/ideval/data/>
- [29] T. M. Mitchell, *Machine Learning*, 1st ed. New York, NY, USA: McGraw-Hill, Inc., 1997.

- [30] A. Moore, D. Zuev, and M. Crogan, *Discriminators for use in flow-based classification*. Queen Mary and Westfield College, Department of Computer Science, 2005.
- [31] A. W. Moore and K. Papagiannaki, “Toward the accurate identification of network applications,” in *Passive and Active Network Measurement*. Springer, 2005, pp. 41–54.
- [32] B. E. Mullins, T. H. Lacey, R. F. Mills, J. M. Trechter, and S. D. Bass, “How the cyber defense exercise shaped an information-assurance curriculum,” *Security & Privacy, IEEE*, vol. 5, no. 5, pp. 40–49, 2007.
- [33] T. T. Nguyen and G. Armitage, “A survey of techniques for internet traffic classification using machine learning,” *Communications Surveys & Tutorials, IEEE*, vol. 10, no. 4, pp. 56–76, 2008.
- [34] V. Paxson, “Bro: a system for detecting network intruders in real-time,” *Computer networks*, vol. 31, no. 23, pp. 2435–2463, 1999.
- [35] J. C. Platt, “Fast training of support vector machines using sequential minimal optimization,” in *Advances in kernel methods*. MIT press, 1999, pp. 185–208.
- [36] J. R. Quinlan, “Improved use of continuous attributes in C4.5,” *arXiv preprint cs/9603103*, 1996.
- [37] —, *C4.5: programs for machine learning*. Morgan kaufmann, 1993, vol. 1.
- [38] M. Roesch *et al.*, “Snort: Lightweight intrusion detection for networks.” in *LISA*, vol. 99, 1999, pp. 229–238.
- [39] C. Sanders and J. Smith, *Applied Network Security Monitoring: Collection, Detection, and Analysis*, 1st ed. Syngress Publishing, 2013.
- [40] K. A. Scarfone and P. M. Mell, “SP 800-94. Guide to Intrusion Detection and Prevention Systems (IDPS),” 2007.
- [41] R. Sommer and V. Paxson, “Outside the closed world: On using machine learning for network intrusion detection,” in *Security and Privacy (SP), 2010 IEEE Symposium on*. IEEE, 2010, pp. 305–316.
- [42] J. Song, H. Takakura, Y. Okabe, M. Eto, D. Inoue, and K. Nakao, “Statistical analysis of honeypot data and building of Kyoto 2006+ dataset for NIDS evaluation,” in *Proceedings of the First Workshop on Building Analysis Datasets and Gathering Experience Returns for Security*. ACM, 2011, pp. 29–36.
- [43] C. T. Symons and J. M. Beaver, “Nonparametric semi-supervised learning for network intrusion detection: combining performance improvements with realistic in-situ training,” in *Proceedings of the 5th ACM workshop on Security and artificial intelligence*. ACM, 2012, pp. 49–58.

- [44] E. Tombini, H. Debar, L. Mé, and M. Ducassé, “A serial combination of anomaly and misuse IDSes applied to HTTP traffic,” in *Computer Security Applications Conference, 2004. 20th Annual.* IEEE, 2004, pp. 428–437.
- [45] Verizon, “2013 Data Breach Investigations Report,” 2013. [Online]. Available: http://www.verizonenterprise.com/resources/reports/rp-data-breach-investigations-report-2013_en_xg.pdf
- [46] I. H. Witten, E. Frank, and A. Mark, *Data Mining: Practical machine learning tools and techniques*, 3rd ed. Morgan Kaufmann, San Francisco. Retrieved, 2011.
- [47] D. H. Wolpert and W. G. Macready, “No free lunch theorems for optimization,” *Evolutionary Computation, IEEE Transactions on*, vol. 1, no. 1, pp. 67–82, 1997.
- [48] J. Zhang and M. Zulkernine, “A hybrid network intrusion detection technique using random forests,” in *Availability, Reliability and Security, 2006. ARES 2006. The First International Conference on.* IEEE, 2006, pp. 8–pp.
- [49] J. Zhang, M. Zulkernine, and A. Haque, “Random-forests-based network intrusion detection systems,” *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 38, no. 5, pp. 649–659, 2008.

REPORT DOCUMENTATION PAGE					<i>Form Approved</i> <i>OMB No. 0704-0188</i>	
The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.						
1. REPORT DATE (<i>DD-MM-YYYY</i>)		2. REPORT TYPE		3. DATES COVERED (<i>From — To</i>)		
26-03-2015		Master's Thesis		June 2013 — Mar 2015		
4. TITLE AND SUBTITLE				5a. CONTRACT NUMBER		
Evaluating Machine Learning Classifiers for Hybrid Network Intrusion Detection Systems				5b. GRANT NUMBER		
				5c. PROGRAM ELEMENT NUMBER		
6. AUTHOR(S)				5d. PROJECT NUMBER		
Rich, Michael D., Master Sergeant, USAF				N/A		
				5e. TASK NUMBER		
				5f. WORK UNIT NUMBER		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)				8. PERFORMING ORGANIZATION REPORT NUMBER		
Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 Hobson Way WPAFB OH 45433-7765				AFIT-ENG-MS-15-M-046		
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)				10. SPONSOR/MONITOR'S ACRONYM(S)		
AFRL Sensors & AF Cyberspace Technical Center of Excellence Attn: Dr. Steven K. Rogers 2241 Avionics Circle, Bldg 600 WPAFB OH 45433-7765 (937) 528-8525, steven.rogers@us.af.mil				AFRL/RV & AF CyTCoE		
				11. SPONSOR/MONITOR'S REPORT NUMBER(S)		
12. DISTRIBUTION / AVAILABILITY STATEMENT						
DISTRIBUTION STATEMENT A: APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.						
13. SUPPLEMENTARY NOTES						
This work is declared a work of the U.S. Government and is not subject to copyright protection in the United States.						
14. ABSTRACT						
Existing classifier evaluation methods do not fully capture the intended use of classifiers in hybrid intrusion detection systems (IDS), systems that employ machine learning alongside a signature-based IDS. This research challenges traditional classifier evaluation methods in favor of a value-focused evaluation method that incorporates evaluator-specific weights for classifier and prediction threshold selection. By allowing the evaluator to weight known and unknown threat detection by alert classification, classifier selection is optimized to evaluator values for this application. The proposed evaluation methods are applied to a Cyber Defense Exercise (CDX) dataset. Network data is processed to produce connection-level features, then labeled using packet-level alerts from a signature-based IDS. Seven machine learning algorithms are evaluated using traditional methods and the value-focused method. Comparing results demonstrates fallacies with traditional methods that do not consider evaluator values. Classifier selection fallacies are revealed in 2 of 5 notional weighting schemes and prediction threshold selection fallacies are revealed in 5 of 5 weighting schemes.						
15. SUBJECT TERMS						
Intrusion Detection Systems, Machine Learning, Threat Classification, Anomaly Detection						
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON	
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. Robert F. Mills, AFIT/ENG	
U	U	U	U	131	19b. TELEPHONE NUMBER (<i>include area code</i>) (937) 255-3636 ext 4527; robert.mills@afit.edu	