

12-1-2012

Methods for speculatively bootstrapping better intrusion detection system performance

Sunny Fugate

Follow this and additional works at: https://digitalrepository.unm.edu/cs_etds

Recommended Citation

Fugate, Sunny. "Methods for speculatively bootstrapping better intrusion detection system performance." (2012).
https://digitalrepository.unm.edu/cs_etds/20

This Dissertation is brought to you for free and open access by the Engineering ETDs at UNM Digital Repository. It has been accepted for inclusion in Computer Science ETDs by an authorized administrator of UNM Digital Repository. For more information, please contact disc@unm.edu.

Sunny James Fugate

Candidate

Computer Science

Department

This dissertation is approved, and it is acceptable in quality and form for publication:

Approved by the Dissertation Committee:

George F. Luger , Chairperson

Thomas P. Hayes

Jedidiah R. Crandall

Thomas P. Caudell

LorRaine T. Duffy

Methods for Speculatively Bootstrapping Better Intrusion Detection System Performance

by

Sunny James Fugate

B.S., Electrical Engineering, University of Nevada, 2002
M.S., Computer Science, University of New Mexico, 2011

DISSERTATION

Submitted in Partial Fulfillment of the
Requirements for the Degree of

**Doctor of Philosophy
Computer Science**

The University of New Mexico

Albuquerque, New Mexico

December 2012

©2012, Sunny James Fugate

Dedication

*To my brilliant wife Truc-Ha, my sons Minh and Maxwell,
and my friends and colleagues who have supported and encouraged me.*

Acknowledgments

I would foremost like to thank my adviser, Professor George Luger, for supporting my research even though it often drifted from his own interests. My time at the University of New Mexico would not have been nearly as pleasurable without your insights, patience, and candor (and occasional infusions of India Pale Ale at the local brew pub).

As a recipient of the Naval Postgraduate School's Science, Mathematics, & Research for Transformation (SMART) Scholarship, I am eternally grateful for the financial means to obtain an advanced degree while maintaining my pursuit of Navy-directed research as a government civilian. The financial freedom afforded by the SMART program has enabled me to start a family and learn about human cognition and development as both a scientist and a parent.

I would also like to thank the Defense Threat Reduction Agency for supporting my application to the SMART program, the, Systems Center (SPAWAR) for their continued employment throughout my degree studies, and the Office of Naval Research (ONR) for funding my research projects throughout my studies. Many aspects of this dissertation were partially funded through ONR.

Dr. LorRaine Duffy has been my mentor, adviser, and friend during my first decade of government service. She has been my unequivocal advocate throughout both my academic studies and research efforts. Under her guidance, I have had the most fun, worked with the best teams, met the most fascinating people, and worked on the most clever ideas of my early career. Thank you for believing in me, trusting me, and being my vocal advocate in a forest of falling trees. Someday, the rest of the guys and I will make you famous, just you wait.

Finally, I would like to thank Christine Lisman. Without her advocacy of my early efforts in security-related visualization (and my having subsequently followed her out to Washington, DC to work at the Joint Task Force for Global Network Operations) I may have never gotten my feet wet with Network Intrusion Detection in the first place.

Methods for Speculatively Bootstrapping Better Intrusion Detection System Performance

by

Sunny James Fugate

B.S., Electrical Engineering, University of Nevada, 2002

M.S., Computer Science, University of New Mexico, 2011

Ph.D., Computer Science, University of New Mexico, 2012

Abstract

During the last three decades, the designers of computer Intrusion Detection System (IDS) have been continuously challenged with performance bottlenecks and scalability issues. The number of threats is enormous. The performance of IDS systems depends primarily on the quantity of input data and complexity of detected patterns. During noisy attacks, system load tends to increase proportional to increasing data rates, making IDS systems vulnerable to flooding and denial-of-service attacks. Unfortunately, the number, type, and sophistication of threats is quickly increasing, outpacing our ability to detect them. The more we try to detect, the more computing and economic resources must be reserved solely for the task of detection, whittling away at what remains for performing useful computations.

This dissertation describes methods for assessing the current scaling performance of signature-based IDS and presents models for speculatively bootstrapping better IDS performance. Using measurements of the coverage and scaling performance of a modern signature-based IDS in the context of an anticipatory model arguments are presented that

maintaining compact, low-coverage signature-sets does not provide optimal protection for modern heterogeneous computing environments. The primary contribution is an analysis of how mechanisms of anticipatory bias can be used to achieve performance improvements.

To support the theoretical models, two principal approaches have been implemented. The first uses a combination of anticipation and feedback in an attempt to decrease per-signature costs by (counter-intuitively) increasing system coverage. The approach uses learned sequence statistics to make predictions of future events. Each prediction above a chosen threshold is used to decrease per-stream detection cost by shunting traffic to smaller detectors (at the risk of increased error rates). The new approach promises decreasing per-signature costs as new detection signatures are added. The design and performance of a prototype anticipatory IDS, “Packet Wrangler”, demonstrates the feasibility of the basic approach.

The second approach applies primarily to improving the performance of IDS when under stress. When overburdened, an IDS will drop input data (often arbitrarily). A *probabilistic signature activation* approach is described which improves error rates by decreasing the total amount of input data lost by probabilistically dropping signature activations based on learned event statistics and system load. A theoretical analysis is presented which shows that a policy which drops signatures instead of packets can outperform the default policy of dropping packets in terms of total error rates. A rudimentary prototype based on the Snort IDS, “Probabilistic Flowbits”, is described. Experimental results are then given which show substantially decreased error rates while simultaneously decreasing system overhead.

In conclusion, a case is made for expanding IDS coverage and implementation fast-feedback and anticipatory optimizations. It can be argued that these extensions are both necessary and sufficient for long-term scalability, but oddly absent from existing systems.

Contents

Abstract	vi
Figures	xii
Tables	xiv
Listings	xvi
1 Introduction	1
1.1 Summary of Results	5
2 Background	10
2.1 Why Study Anticipatory Intrusion Detection?	10
2.1.1 Inspiration	10
2.1.2 Software Complexity, Vulnerability, and Detection	13
2.1.3 The Problem of Performance	15
2.1.4 Goals	17
2.2 Intrusion Detection Systems	21
2.2.1 Host versus Network Detection	22
2.2.2 Misuse and Anomaly Detection	24
2.2.3 Signature-Based Detection	26
2.3 Characteristics of Signature-Based Detection	29
2.3.1 Decision Procedure	29
2.3.2 Signature Information	32
2.3.3 Alert & Signature Semantics	33
2.3.4 Traffic Patterns	36

CONTENTS

2.4	Per-Signature Cost Optimization	37
2.4.1	Flow Tracking	38
2.4.2	Signature Chaining	38
2.4.3	Event Filtering & Suppression	39
2.5	Related Work	40
2.5.1	Attacker Modeling	42
2.5.2	Prediction & Performance Adaptation	44
2.5.3	Historical Surveys	49
3	Theory	54
3.1	An Anticipatory IDS Model	55
3.1.1	Wasted Information	58
3.1.2	Threat Coverage	63
3.1.3	Packet Coverage	64
3.1.4	Signature Equivalence Classes	66
3.1.5	An IDS Cost Function	68
3.2	Probabilistic Signature Activation	70
3.2.1	Signature Activation Policies	71
3.2.2	Probabilistic Flowbits and False Negatives	75
3.3	Predictor Errors	83
3.3.1	Signature Chaining & Flow Tracking	83
3.3.2	Systematic Errors	84
3.3.3	Error Detection	85
3.4	The Detection Game	86
4	IDS Performance Characteristics	90
4.1	Generating Large Signature-Sets	91
4.2	Snort IDS Performance Scaling	93
4.2.1	Startup Performance	93

CONTENTS

4.2.2	Packet Coverage	94
4.2.3	CPU-time scaling	96
4.2.4	Packet Loss Scaling Performance	97
4.2.5	Anticipatory Gain Potential of the Snort IDS	101
5	Prototypes	105
5.1	Packet Wrangler	105
5.1.1	Startup	106
5.1.2	Program Logic	108
5.1.3	Predictors	109
5.1.4	Discussion	111
5.2	Probabilistic Flowbits	113
6	Experiments	117
6.1	Methods	117
6.2	Packet Wrangler	119
6.2.1	Naive Bayes Predictor	120
6.2.2	Rule-Based	125
6.2.3	Traffic Split	129
6.2.4	Discussion	131
6.3	Probabilistic Flowbits	133
6.3.1	Results	134
6.3.2	Analysis & Criticisms	141
7	Impact, Conclusions, & Future Work	144
7.1	A “Household Survey” Analogy	146
7.2	Biological Metaphors & Future Work	148
A	Hardware & Software Configurations	156
A.1	Test System Configuration	156

CONTENTS

A.2 Training and Test Data	158
B Snort Configurations & Analysis	164
B.1 Basic Snort Configuration File	164
B.2 Snort Signature Profiling & Tuning	165
B.3 Custom Snort Signatures	170
C Issues & Confounding Factors	172
C.1 Event Suppression	172
C.2 Packet Loss	173
D Scripts & Algorithms	176
D.1 Packet Wrangler	176
D.2 Gain Curve Calculations	177
References	180

Figures

2.1	Software growth over a decade of Linux operating system development . . .	13
2.2	Cataloged vulnerabilities per calendar year	14
2.3	Signature-set processing times for the Snort IDS release 2.9.1.2	16
2.4	Temporal logic relations over IDS events	36
2.5	Temporal logic using Next(alert) semantics	36
3.1	Predictive IDS architecture denoting conventional techniques	56
3.2	Dual decision tree representations with color-labeled equivalence classes .	57
3.3	Anticipatory IDS architecture	58
3.4	Probabilistic signature activation function	74
3.5	Simple state transitions for Flowbit-based signature activation	75
3.6	The Detection Game payoff matrix for a zero-sum game	87
4.1	Random feature replacement algorithm	92
4.2	Polynomial fit for the startup time of the Snort IDS	94
4.3	Linear scaling of total CPU-time with signature-set size of the Snort IDS .	96
4.4	Test 18-2: Packets dropped by Snort for signature-set sizes 1K-40K . . .	98
4.5	Test 18-1: Snort scaling for signature-set sizes from 1K-40K	99
4.6	Over-estimating fit for packet processing rate	100
4.7	Packet processing rate estimate curves	100
4.8	Curves of constant gain for signature-set size of $n = 4320$	102
4.9	Curves of constant gain for signature-set size of $n = 100000$	103

FIGURES

5.1 Predictive IDS architecture showing data flow 106

5.2 Flow diagram for the Packet Wrangler program 108

6.1 Test 5: Packet Wrangler forwarding using the *Top 50* event sequences . . . 121

6.2 Test 15: Packet Wrangler Top 50 Configuration 124

6.3 Test 36: Process Times showing experimental overhead costs 131

6.4 Test 36: Process Times with event filter 132

6.5 Test 38-1b: Total CPU-time for all processes (0min-10min window) . . . 135

6.6 Test 38-2b: Total CPU-time for all processes (10min-20min window) . . . 136

A.1 Stochastic matrix set showing Snort’s built-in preprocessors 161

A.2 Example stochastic matrix using the entire set of available signatures . . . 162

B.1 A Snort signature-set profile of top 10 signatures (Avg/Check) 166

Tables

2.1	Linear temporal logic relations	35
2.2	Intrusion Detection Systems	50
3.1	Equivalence class size and problem size	68
6.1	Test 29a,b: HTTP traffic split [s1 flow tracking, s1 HTTP-detection] . . .	127
6.2	Test 29c: HTTP traffic split [no s1 flow tracking, 3-tuple]	128
6.3	Tests 30-32: [small secondary signature-sets]	129
6.4	Test 35h: Dropping all detected TCP packets for a high-load scenario . . .	130
6.5	Test 36: Measuring cost of I/O for traffic detection signatures	131
6.6	Test 38-1b: packets processed and error rates (0min-10min)	135
6.7	Tests 38-1b & 2b: Cost-relevance policy relative to signature 4677	136
6.8	Test 38-2b: Packet processing and error rates (10min-20min window) . .	137
6.9	Test 38-2j: Cost-relevance policy with re-scaled probabilities	138
6.10	Test 38-2j: Packet processing and error rates after signature removal . . .	138
6.11	Calculated gains for various signature activation policies	139
6.12	Test 38-2k: Cost policy probability table	140
6.13	Test 38-2k: Packet processing and error rates using a Cost policy	141
A.1	System Hardware Configuration	157
A.2	System Software Configuration	157
A.3	High Occurrence Alert Sequences for the 1998 Dataset	160
B.1	Top 10 signatures in terms of total CPU-time in microseconds	168

TABLES

B.2	Top 10 signatures in terms of total CPU-time using a Cost policy	169
B.3	Cost policy performance differences in total CPU-time and checks	169

Listings

2.1	Example of a Snort Signature	34
2.2	Predicate logic representation of a Snort Signature	34
A.3	Snort configuration options used for compilation	156
A.4	Snort command using the PF_RING data acquisition module	157
A.5	Snort command for experiments	158
A.6	TCPReplay command	158
B.7	High-cost signature 4677 reproduced from a Snort IDS signature-set	166
B.8	Simple HTTP activity detection signatures	170
B.9	TCP Detection Signatures	171
D.10	Packet Wrangler: predictEquivalenceClasses(Alert Set A_i , Connection C_i)	176
D.11	Packet Wrangler: updateAllFilters	176
D.12	Packet Wrangler: updateFilter(Connection C_i , Equivalence classes E_i)	176
D.13	Setup for gain curve computation in Mathematica	178
D.14	Gain curves computation for $n=4320$, G in $\{1,10,20,\dots,100\}$	178

LISTINGS

D.15 Gain curves computation for $n=100000$, G in $\{1,10,20,\dots,100\}$ 178

Chapter 1

Introduction

Each and every day, our bodies wander through a morass of other organisms, proteins, and matter. They assault us via the front lines of skin, lungs, and gut. The total exposed surface area of around $400m^2$ is truly extraordinary¹. Via evolution, our bodies have learned to process a great diversity of substances (air, water, foods, toxins, casual contacts), all while efficiently identifying and filtering pathogenic material. The cost of not detecting a threat is disease, possibly death. The cost of overreacting is allergy, asthma, and possibly anaphylactic shock. Each day the body strives for the happy median in between – all without conscious involvement.

During the last few decades we have created a rich ecology of sophisticated computing machines. Each day these devices are more and more extensions of our bodies and our minds. We gain great advantage through their use, but we have exposed ourselves in ways unimaginable by any process of natural evolution. We have expanded our total surface area by inextricably tying ourselves into the realm of information and networks. Somehow, however, we never realized the extent of our newfound vulnerabilities. We protect our digital surface areas using a bubble-boy approach to preventing disease. Our protection

¹<http://www.vendian.org/envelope/dir2/lungsout.html> (Webpage - Retrieved Aug. 12, 2012)

Chapter 1. Introduction

mechanisms are complex barriers, detection, and filtering mechanisms. We attempt to limit our exposure, but any exposure is enough to destroy us. Just as for the premature child in an intensive-care unit, regimented isolation is not a permanent solution. The costs are debilitating. The risks do not dissipate the longer we wait.

In this thesis, we examine a particular type of defense mechanism used for detecting computational pathogens, the Intrusion Detection System or IDS.

During the last 25 years, IDS designers have been continuously challenged with performance bottlenecks and scalability issues. The number of threats is enormous. But the more we try to detect, the more of our computing resources are being used solely for protection, whittling away at what remains for performing useful computations. The performance of many IDS systems depends primarily on the quantity of input data and complexity of detected patterns. During noisy attacks, system load tends to increase proportional to increasing data rates, making IDS systems vulnerable to flooding and denial-of-service attacks. Unfortunately, the number, type, and sophistication of threats is quickly increasing, outpacing our ability to detect them. Anyone who has run a background anti-virus program on their computer can attest to the impact such systems have on our productivity.

The research community has met these challenges with a sophisticated arsenal of clever algorithms for achieving low-cost pattern-matching, decreasing per-packet (and per-signature) detection costs. However, there has been little work in assessing whether traditional signature-based IDS approaches can achieve long-term scalability in the face of increasingly multifarious threats.

Our desire is to *cheaply detect* all of the relevant threats for our computer networks. The current problem is that these systems scale poorly. Detecting more things requires more hardware, so much so that nobody (except maybe the US Department of Defense) attempts to detect everything. Parallelization, often touted as the solution to all scalability problems, is really just more hardware in a smaller space. It is part of the solution, but it does not

Chapter 1. Introduction

make our algorithms smarter nor does it significantly change the scaling performance of our existing detection systems. These systems all scale proportional to the size of the detection task. The larger the detection task, the larger the computing and storage costs. This is unacceptable.

However, precise detection does not have to cost us so much. Instead of simply being a thorn in our side, precise detection should be a means to an end in respect to scalability. But systems must be designed to take advantage of it. The principle thesis of this dissertation is that detection systems can achieve better scaling performance by using increased coverage to anticipate future events.

Currently, detection system performance proportionally degrades as coverage is increased, but if the added coverage allows some events to be easily predicted, increasing detection system coverage can make overall detection less expensive. The compromise is that each bit of information gain must be achieved with minimum cost, more information must be retained over time, wasted information must be eliminated whenever possible, and the detection system must be made more sophisticated through the use of closed-loop feedback for online detector optimization.

Signature-based intrusion detection systems utilize large collections of fixed patterns to detect malicious events on computers and computer networks. These systems are sometimes associated with single computing system, as in a Host Intrusion Detection System (HIDS). They may also sit at a router or firewall and examine traffic from a network of computing systems, as in a Network Intrusion Detection System (NIDS).

In brief, a “signature” is a collection of features which can be used to uniquely identify a specific malicious event (see Sect. 2.2.3 on page 26). The number of potential signatures necessary to precisely identify all current threats is daunting. As of May 2012, MITRE Corporation’s Common Vulnerabilities and Exposures (CVE) database has cataloged approximately 55,000 *disclosed* vulnerabilities dating from 1999. This only accounts for a

Chapter 1. Introduction

fraction of the exploits available to a well-funded adversary. Indeed the number of malicious programs which might be gainfully identified using signature-based approaches while in-transit over a network number in the millions. Clearly, modern IDS achieve only partial coverage of known vulnerabilities, exploits, and other threat-related activities.

To improve coverage we might simply abandon signature-based approaches and use something wildly different. Statistical anomaly detection approaches provide much better coverage of both current and potential threats. Unfortunately, these types of system do not provide much in terms of precision. They tell us *that* something bad is happening, but not *what* it is or *why* we should do something about it. We have chosen to focus on signature-based approaches because of their ability to precisely identify specific threats. Anomaly detection systems are important, but nobody wants a system that only ever tells them “something bad is happening, but I do not know what it is and I do not know why I feel this way.” We are over-simplifying in order to make a point. We do need to be able to detect statistical anomalies so that we can learn about new threats that we cannot yet identify. But we need precision in order to act and respond quickly and appropriately.

Within modern IDS, most of the information which *could* be learned (and later used) in the process of detecting threats is generally wasted, discarded due to the high cost of processing and storage. IDS tend to be vulnerability focused and are designed to “forget” anything that is not directly identified as a threat. Unfortunately, these forgotten features are exactly what we need to improve the performance of future detection tasks. By throwing away so much information for purposes of efficiency, we preclude an important class of performance optimizations: anticipation. We end up with detection systems which lack sufficient coverage of input features to perform gainful anticipation of future events. This lack of broad-spectrum coverage (and subsequent lack of anticipatory mechanisms) is partially to blame for our current performance bottlenecks and scalability issues.

While the algorithms and approaches of modern IDS are undeniably sophisticated, few have made use of the efficiency tricks and shortcuts commonplace within biological

Chapter 1. Introduction

cognitive and neural systems. Numerous behavioral psychology concepts seem relevant: priming, anticipation, habituation, sensitization, desensitization, associative learning, imprinting. There are also cellular and multicellular dynamics processes that seem particularly fitting, namely: neuronal action potential dynamics such as signal transduction, frequency summation, afterhyperpolarization, and refractory periods; cardiac pacemaker potentials; lateral inhibition between retina horizontal cells; and neurotransmitter-mediated signaling and neuronal dynamics.

We do not need to be experts to be fascinated by these concepts. In particular, we can readily note their apparent applicability (and presence) within an incredibly wide range of phenomenon. We can use these concepts as inspiration for various optimization approaches for intrusion detection systems. While implementations fall short of their biological equivalents, the reader should be inspired to approach detection systems from new perspectives. Nature cheats, takes shortcuts, and takes chances to achieve its sophistication and efficiency. So should we in the design and construction of detection systems.

In summary, we have taken a single, widely used IDS system and applied some of these concepts, namely habituation, desensitization, and priming. Throughout this dissertation we explore the design and performance of IDS (Chapt. 2 and Chapt. 4), craft theoretical models of anticipation and inhibition within detection systems (Chapt. 3), create several prototypes (Chapt. 5), run many experiments, and generate some results (Chapt. 6). While these explorations have not always resulted in the dramatic new insights, they do suggest new perspectives on how detection systems *should* work.

1.1 Summary of Results

The long-term goal of this research is to improve the performance and coverage of signature-based IDS in general. It is hoped that with the right architecture and mechanisms per-

Chapter 1. Introduction

signature cost can be decreased as new signatures are added to such a system. The resulting system would have *strong sub-linear scaling*. While the prototype systems implemented thus far fall short of this goal, it is clear that the direction for future systems is to more intelligently leverage information gained in performing detection.

Two fundamentally different prototypes have been constructed. The first, *Packet Wrangler*, uses a predictor and traffic management mechanism which is wholly separate from the IDS. This prototype was inspired by biological anticipation mechanisms such as priming. The goal is to guess future events and use only the relevant portion of the detection engine. The Packet Wrangler prototype predicts future events and forwards portions of traffic to IDS instances configured with smaller collections of signatures. The prototype is described in Chapt. 5.1. Several experiments and results are shown in Sect. 6.2. In brief, any performance gains were generally small due to high costs of Input/output (I/O) overhead for signature-sets with sufficient coverage. In other words, signatures need to produce information for large portions of the input data. However, the performance gains achievable appear to be proportional to the increase in I/O overhead, generally resulting in a net loss. An analysis of the failure of this prototype to achieve acceptable gains is given in Sect. 6.2.4.

The second prototype, *Probabilistic Flowbits*, performs anticipation in a simpler sense, operates internal to the IDS, operates on individual traffic flows, and thus resolves a number of issues with the Packet Wrangler prototype. In order to achieve appreciable gains it was necessary to eliminate the I/O and processing overhead of the Packet Wrangler approach. The overall approach is inspired by inhibitory feedback mechanisms in biological systems. The prototype is a modification of the internal flow-tracking mechanism of the Snort IDS² to set and check “flowbits” according to a table of pre-configured probabilities. In this way the anticipatory mechanism is simply a per-flow biased sampling of high-cost or high-occurrence signatures with an optimal threshold which depends on system load.³

²<http://snort.org> - *Snort Home Page* (Webpage - Retrieved Aug 2012)

³This follows prior work on adaptive IDS reconfiguration described in Chapt. 2 on page 44.

Chapter 1. Introduction

In brief, we have had partial success in achieving performance gains. Using standard signature sets the Packet Wrangler prototype generally increases system overhead between 4% and 30%. The lack of performance gains is due to a combination of factors: low gains between signature-sets of similar sizes; significant I/O overhead for high-coverage signature-sets; and difficulty in manually crafting high-coverage signatures. The Probabilistic Flowbits prototype addresses many of the issues in the Packet Wrangler approach and produces substantial gains between 20% and 40%. Section 6.3 describes the result of experiments using this system. Sections 6.3.2 presents an analysis of the results and presents potential extensions.

In summary this research has produced the following contributions:

- Presented a method and supporting theory to improve the practical coverage and performance of signature-based IDS.
- Demonstrated improvements to the practical coverage and precision of signature-based IDS.
- Demonstrated improvement of the scaling performance of signature-based IDS.
- Achieved lower per-packet computing cost using anticipatory mechanisms.

Secondarily, we have also explored the following:

- Demonstrated a method for generating large sets of random signatures.
- Explored the scaling performance of Snort IDS for large signature-sets.
- Provided sound justification and means for ignoring sets of signatures for using anticipation and bias.
- Explored the resulting trade-offs between anticipation, expanded coverage, improved performance, and error rates.

Chapter 1. Introduction

- Explored how this approach can be extended and utilized in more widespread applications.

A primary theoretical result is an analytical determination of the efficiency and quality of a signature-based detection system with and without prediction under optimal conditions. An exploration of the theory of anticipation for intrusion detection is important for several reasons. There is currently a wide array of solutions provided by industry, but there are no known systems (or published methods) which take the signature-based system to the logical extreme of a single signature for each exploit and vulnerability. This dissertation attempts to directly address the need for progressively larger signature-sets that has arisen as the software ecosystem has evolved. Engineering solutions abound, but do not address the extreme case where the number of signatures is essentially unbounded. The approach described also calls for further research and application to new systems, such as bio-detection and chemical sensing where the number of entities which should be detected precisely far outnumber the currently tractable detection mechanisms.

In addition to analytic results, this thesis has been supported using experimental results demonstrating the performance of a system which uses anticipation mechanisms. Although some of the results have been limited in scope, the proposed methods promise to enable functioning IDS even in the face of incalculably large decision procedures[60] or (equivalently) severely limited computing power.

To summarize, methods and analysis have been demonstrated which show better scaling performance than perceptually “naked” systems. The extent of this improvement is a principal question of this research. Although we explore both experimental and analytical results on potential performance gains, future research should be capable of greatly extending these findings. Finally, very different types of IDS should be able to benefit using an anticipatory approach. Where Snort is essentially a stateful string-based pattern matcher, the Bro IDS⁴ utilizes a number of anomaly detection methods and incorporates a stateful analysis of

⁴<http://bro-ids.org> - *The Bro Network Security Monitor* (Webpage - Retrieved Aug 2012)

Chapter 1. Introduction

event sequences as specified using a scripting language [72, 78]. Both of these systems, and others should be amenable to augmentation using anticipatory mechanisms, but this is left to future research.

Chapter 2

Background

2.1 Why Study Anticipatory Intrusion Detection?

2.1.1 Inspiration

A primary inspiration of this research is the observation that human beings and other organisms are able to perceive and process a huge number of different stimuli without relying on brute-force search or rote pattern matching. The mechanisms employed by almost any known organism dwarfs that of our most sophisticated computing systems. It is a premise of this research that such organisms must reason abductively about almost everything that is perceived, that without leaping to conclusions first and checking logical supports later, perception is nigh impossible.

Even the simplest organism utilizes multiple knowledge representations and myriad predictive mechanisms tied to the fact-checking apparatus of logic. All organisms are zealots of prediction and inductive bias. Without such bias, no organism (whether a single-celled protozoan or the most intelligent of human beings) would be capable of perceiving much at all. As far as we know, *guessing* is necessary for practical perception.

Chapter 2. Background

The detection of threats is one of every living organisms highest perceptual priorities. This prioritization is an emergent phenomena of our evolution within an often hostile environment. However, if we were to examine the perceptual apparatus of any organism we would most likely find that the detection of threats is balanced carefully and cleverly with the perception of other necessities, such as the location of food, friends, and shelter. Indeed, at least in humans, perception of threats appears to prioritize attention, not to capture it[69]. We might also find that the organism spent an inordinate amount of time perceiving things within its environment that were not directly helpful at all. What a waste.

It might not be obvious, but organisms perceive objects in the world because they can, not because they should. Another way of stating this is that organisms perceive objects *because they are convenient and not because they are necessary*. We are opportunistic and lazy, although an organism well adapted to its environment will perceive what it needs to perceive often enough to survive. The point being made is that most often *what is perceived is what is expected and what is convenient*. This is not necessarily what is needed, even when it is in right front of us.

When our expectations (not our perception!) indicate a threat or are violated, we are able to respond immediately, without delay to reciprocate proactively to an anticipated threat, and reactively to our violated expectations. Our proactive response is a prediction of future precepts, whether it be the taste of a spoon full of strawberry ice-cream or the deadly grasp of a predator. What we currently perceive immediately informs us of what we might perceive next. This is the key to efficient, or even remotely tractable perception. We generally do not need to search for claw wielding predators when eating ice cream, and we certainly do not look for ice cream when escaping a predator. Our environment and prior precepts informs our current means of perception.

Without these perceptual predictions, there are simply too many possible precepts at any given instant. If somehow we could isolate individual objects pre-perceptually, there are often so many in our immediate environment that we it would be difficult to even count

Chapter 2. Background

them. Nonetheless, when our predictions fail we use logic to determine the cause of our failure and an adequate logical resolution. For complex organisms and sufficient time, when our expectations do not match our perception, we perform some form of cognitive reconciliation or behavioral adaptation. Only then do we take the time to manually refine our perceptual predictors using any type of brute-force approach.

Our software detection mechanisms are constructed to be as efficient as possible given our knowledge of algorithms and efficient search. We've been fortunate so far in that enumeration of such things in a brute force manner is not always intractable. But even now there are simply too many things to be perceived in each instant of computing time. This leaves us to produce a predetermined static prioritization of precepts, perhaps optimal in the case when we lack prior knowledge, but rarely optimal in context. Similar to organism cognition, our software defensive mechanisms should first detect what is either the most relevant (based on priors) or the most easily perceived, using this to bias future expectations and the search for explanations. The methods employed must be opportunistic in the same way that our human perceptual mechanisms are opportunistic.

Counter-intuitively, there are generally too many threats to search only for threats. Rather, perceive the immediate, and let this inform more costly and precise mechanisms of threat detection. It is not enough that we might compute ourselves out of the problem. There is not enough time in the microseconds between events on a computer network. The answer of what to perceive at each instance must already be known. Of course, preoccupation by strawberry ice-cream does not predict being eaten by a claw wielding predator, nor should it. But gladly, just as in physical environments, these bizarre juxtapositions rarely occur and should be easy to detect using computationally inexpensive statistical anomaly detection techniques when they do.

2.1.2 Software Complexity, Vulnerability, and Detection

The complexity and diversity of software systems continues to evolve at a frenetic pace. Within just the last decade we have seen a hundred-fold increase in the number of software applications distributed with modern operating systems and similarly rapid growth of even formerly nascent computer operating systems such as Linux. Figure 2.1 shows the number of packages in the Debian Linux operating system release and the lines of code within the Linux Kernel from its inception. From its origins in the mid-90's, the Linux operating system kernel has grown from a pet project to representing 10's of millions of source lines of code. Similar growth has been seen in other software applications and development efforts. Measures as simple as the number of source lines of code have long been known to correlate with software defects [52, 83]. Defects occasionally lead to vulnerabilities and vulnerabilities are often exploited.

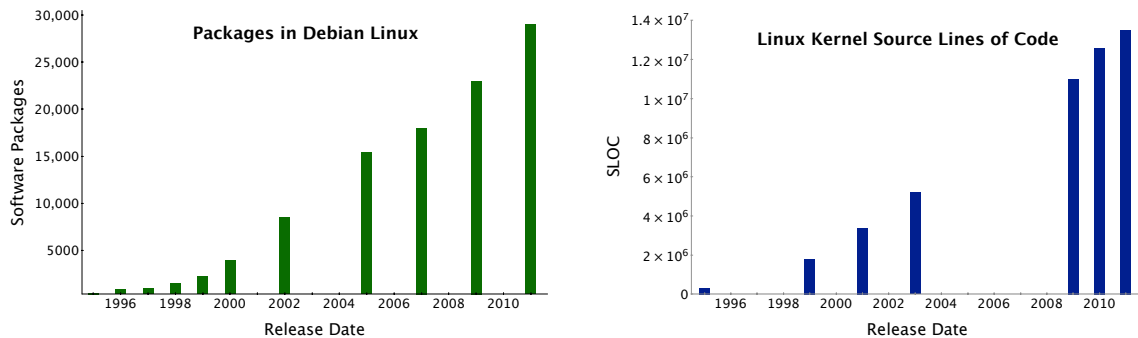


Figure 2.1: Software growth over a decade of Linux operating system development

Source: Wikipedia articles: “Lines Of Code” & “Debian Linux”, Feb 5, 2011

The growth in software size, complexity, and the resulting number of vulnerable defects appears to be growing at an exponential rate. Within the last decade, clever engineering approaches to computer defense have allowed existing technologies and approaches to scale even in the face of an increasing number and complexity of threats. However, the threats covered by those approaches, just as those cataloged by organizations such as Carnegie Mellon’s Computer Emergency Response Team (CERT) (within the Common Vulnerabil-

Chapter 2. Background

ities and Exposures database) represent a fraction of known software vulnerabilities and just a small sample of all of the vulnerabilities within the wild (Figures 2.2). As a result of the enormous growth of software ecologies and latent threats, current approaches to threat detection and response will become increasingly difficult as we extrapolate even a few years.

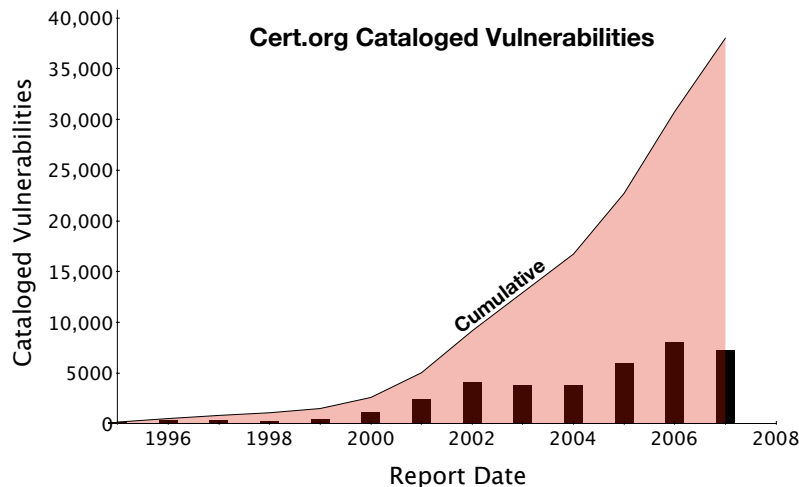


Figure 2.2: Cataloged vulnerabilities per calendar year

Source: Carnegie Mellon - CERT Statistics (Historical): <http://www.cert.org/stats>, Feb 5, 2011

Catalogues of known and disclosed vulnerabilities, however, are not a measure of threat. Common vulnerabilities are generally the target of numerous exploits. Blackmarket exploit packages are likely to contain a large number of current and sophisticated attack tools, many of which we have no way of identifying using IDS. Signature-based detection only works when we have carefully characterized the threat. For network-based IDS we first require enough representative network traffic to generate compact, high-confidence signatures. Internet Protocol (IP) packets cannot be pre-labelled as “bad” or “good” even though we often wish this were the case.¹ Often, the traffic used to create and test new IDS signatures must be manually crafted and may represent difficult to obtain exploits.

¹See RFC 3514, *The Security Flag in the IPv4 Header* - <http://www.ietf.org/rfc/rfc3514.txt>

Chapter 2. Background

As a researcher, the number of readily available exploits represents only a small fraction of the exploits which are written and deployed by adversaries. The well-known “Metasploit” framework includes approximately 800 of such exploits for common vulnerabilities². Penetration testing using such tools is extremely time consuming and generally reserved for high-criticality systems and environments. This is one reason that detection is so desirable. When we cannot prove a particular attack is possible (by attempting it ourselves and taking steps to prevent the attack vector) detection becomes top priority. In many instances and for many organizations, detection may be the only option.

It is clear that computing systems and networks suffer from an ever increasing demand for comprehensive, robust, and efficient detection mechanisms. As the number of software products, capabilities, and resulting vulnerabilities have increased, so has the number and sophistication of attacks. Computing improvements such as faster processors, faster networks, parallelization, increased storage, and faster storage will increase the number and sophistication of attacks just as they improve our ability to detect and respond. Smarter algorithms are needed. Modern approaches represent a comprehensive arsenal of sophisticated defensive tools and techniques, but we are still falling short. Some of these are discussed in Sect. 2.2 on page 21.

2.1.3 The Problem of Performance

Whether or not existing approaches adequately address current threats is a matter of debate. A thorough review of existing literature and familiarity with the commercial capabilities suggests that existing network-based IDS approaches generally lack adequate *coverage* and have linear *cost* scaling (in terms of packets or signatures). Signature-based IDS generally lack coverage of particular exploits and polymorphic variants being generally vulnerability-based rather than exploit-based as a matter of tractability. A more detailed discussion of

²<http://downloads.metasploit.com/data/releases> - *The Metasploit Project* (Retrieved on Aug. 12, 2012)

Chapter 2. Background

some of the issues associated with signature-based IDS are presented in Sect. 2.2.3 on page 26.

A straightforward measurement of the Snort IDS suggests linear scaling even for relatively small numbers of signatures. Figure 2.3 shows the total clock time required to process an identical dataset for various sized signature-set sizes (selected randomly from the pool of signatures distributed in release 2.9.1.2 of the widely available Snort IDS). The outlier at approximately 4300 signatures represent the default signature-set released by the Sourcefire team (which is highly tuned for performance). The official signature release was used for this test vice experimental signature-sets which are readily available online.

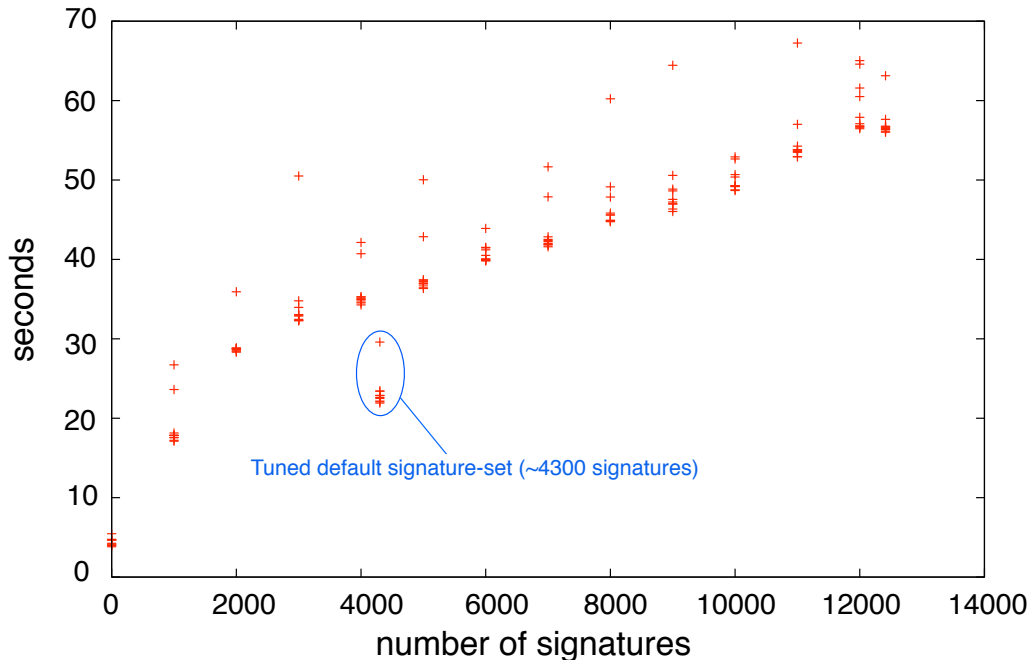


Figure 2.3: Signature-set processing times for the Snort IDS release 2.9.1.2

Ignoring differences in the cost of various signatures (as they are not the focus of this research), at the limit, current techniques require $O(p n)$ comparisons of packets p with

Chapter 2. Background

signatures n . This is true even when clever algorithms are used to construct optimal decision trees over the signature-set (see section 2.3 on page 29).

Certainly, this is linear scaling and not immediately alarming. However, the number of relevant threats is increasing rapidly. The number of necessary signatures is growing as some function of the number of deployed software instances, hardware instances, protocol instances, versions, and vulnerabilities (Fig. 2.2). It seems clear that these systems are not sufficiently *scalable*.

The computing cost of most NIDS is within a constant factor of $O(p n)$. This forces IDS designers to use small sets of broad signatures which focus on vulnerability rather than exploit identification (resulting in poor coverage of exploits, and essentially no coverage of other characteristics). As an example, the Snort IDS (version 2.9.1.2) ships with approximately 6 thousand signatures which are enabled by default. Another 12 thousand signatures are readily available for use, but not enabled due to the significantly increased burden on the system and the potential obsolescence of older signatures. For these reasons and others, significant improvements in the ability to cheaply prioritize signature computations are needed.

2.1.4 Goals

A *scalable* system grows in computing cost (measured in bit-operations) at a rate proportional to the size of its input and task complexity such that the system can continue to function within its original design parameters (e.g. linear or sublinear scaling) without fundamental changes to its design. This research effort has focused on signature-based detection based on our need for better performance and better coverage. An interesting perspective on this problem is that instead of trading performance for coverage we may be able to intelligently leverage improved coverage to directly improve IDS performance. Rather than proportionally degrading (in the best cases), performance of an IDS might be

Chapter 2. Background

made to improve through the addition of appropriate signatures and the use of anticipation and other forms of predictive bias.

More precisely, we can define the relevant aspects of a signature-based IDS as follows:

- A *signature*, s , is a fixed or dynamic pattern (such as a string or regular expression) which can be used by a detector to identify threats and generate output alerts from input data.
- A *signature-set*, S , is a set of signatures which are used in combination within by a detection system.
- An *alert set*, A , is a union of the sets of vulnerabilities A_{vuln} , exploits A_{exploit} , victim characteristics A_{victim} , and attacker characteristics A_{attacker} which are accurately identified by an IDS (i.e. $A = A_{\text{vuln}} \cup A_{\text{exploit}} \cup A_{\text{victim}} \cup A_{\text{attacker}}$).
- The Threat Coverage (TC) measures the portion of possible attacks which can be properly characterized using an alert set.
- *Performance* is defined using the conventional measures of precision, recall, accuracy, and specificity.

The key claims being explored in this research are that signature-based IDS are not currently scalable because they are:

- Imprecise with respect to exploits and tertiary characteristics (such as victim and attacker characteristics), and therefore lack coverage of features sufficient for useful prediction.
- Lack mechanisms which dynamically improve detection system performance based on prior information.

Chapter 2. Background

Given more precise information of an ongoing attack, in many cases we should be able to predict future attacker actions or attack characteristics and to use this information to prioritize future detection computations. The idea is conceptually simple:

By using the prior knowledge of previous detections, resource utilization can be optimized for each network conversation by prioritizing matches of new packets against subsets of the detection tasks, these subsets being based upon a set of anticipated behaviors, attacks, or characteristics.

Traditional IDS techniques perform detection blindly in respect to prior knowledge, with the obvious exception of protocol state-machine tracking (such as Transmission Control Protocol (TCP) state tracking). In the worst-case, a signature-set would be applied sequentially, giving linear Central Processing Unit (CPU)-time performance in the length of packets, signatures, and outputs. In many existing systems optimal decision trees[53] or clever discrete finite-state machines[33] are used in combination with fast string matching algorithms[68, 90]. Many significant internal performance improvements are possible, but most allow for only short-term gains in the face of dramatically expanding threats.

Ignoring differences in the costs of individual feature comparisons, existing detection systems perform an equivalent amount of work for every packet. Using a hypothetical predictive mechanism, signature matching might be more efficiently accomplished by traversing (or selectively ignoring) subsets of the original decision procedure. Such an approach is able to take advantage of recent advances in string matching and regular expression evaluation because it occurs separate from the IDS pattern matching facility. Prediction and anticipatory techniques might be used at many levels within an IDS architecture.

If performed external to the IDS, at an event level, this represents the same semantic level as the attacker's actions. As such, it should be possible to use additional IDS coverage to directly improve performance. If particularly specific signatures are used, in some cases it might be possible to directly choose subsequent signatures without performing decision tree

Chapter 2. Background

traversal. This would dramatically decrease the overall computational cost for individual connections when sufficiently accurate predictions are available.

If performed internal to the IDS, at the granularity of individual packets or signatures, this represents a semantic level of the internal IDS logic and algorithms. A significant amount of work has been done on implementing and improving internal detection mechanisms, some improvements incorporating prediction[90]. Nonetheless, some performance gaps still exist (primarily due to safety and performance assumptions). Probabilistic approaches (generally absent from signature-based detection systems) will likely play a central role in future systems.

Unfortunately, the linchpin of an anticipatory approach appears to be increasing coverage in a way that improves the predictive power of events while limiting increasing I/O and overhead costs. Most NIDS effectively only cover vulnerabilities A_{vuln} , with little or no coverage of exploits or attacker and victim characteristics. Including these in the coverage set could easily expand the number of signatures into hundreds of thousands. However, expanding either the number of signatures or the amount of output has an associated cost. Most IDS are not designed to retain tertiary characteristics. The resulting I/O and memory costs can be substantial. In order for an anticipatory approach to work performance gains must be substantially larger than the increased costs associated with increased coverage.

It should be the case that detection can be performed ever more efficiently as new knowledge is added to a detection system. These types of systems should have strong sublinear scaling in respect to signature-set size. While this research effort has been able to demonstrate limited performance gains for particular configurations with existing signature-sets, the overall research goal is not simply to improve the runtime performance of current IDS approaches. The goal is to increase the practical coverage of these types of systems. In this light we have attempted many techniques for building more efficient detection systems, some which directly address issues of expanded IDS coverage.

2.2 Intrusion Detection Systems

As initially described by Denning in her seminal paper “An Intrusion-Detection Model”[31] and summarized by Bishop[19], IDS can be loosely categorized into *misuse detection* and *anomaly detection*. The difference between these two approaches is essentially one of either characterizing known bad patterns and detecting instances which match the patterns (misuse detection) versus characterizing known good behaviors and detecting deviations (anomaly detection). Denning originally described the pattern matching strategy of signature-based IDS as utilizing both formal and informal specifications and knowledge models such as expert systems. Patterns examined by such systems were also conceived as extending beyond the individual entities being examined (e.g. IP packets, process execution traces, memory accesses) and describe complex patterns of interdependent activity.

One of the most widely used IDS is the signature-based Snort IDS[78]. Others include the event-based BroIDS[72]), *statistical anomaly (behavioral) detection* systems (such as the Time-based Inductive Machine (TIM) system[87]), and *hybrid* systems which combine both approaches. Of particular relevance to this research are Denning’s initial Markov detection model[31] as well as systems which use temporal predictors as embodied by systems such as TIM[87]. In some respects, most existing systems combine many of the aspects of misuse and anomaly detection. Rather than delve into a discussion of the ways in which these different approaches can be implemented, we treat systems as belonging primarily to one or another of either misuse or anomaly detection. The particulars of individual systems not studied in this research effort is left for the reader to explore (see Sect. 2.5 on page 40).

The primary distinction made in this dissertation (relating to to misuse-detection in particular) is a fundamental difference in the use of a predictor for online performance optimization rather than as the detector itself. Our principal inquiry is *whether we can bootstrap a predictor and system performance gains by **increasing** the coverage of the*

Chapter 2. Background

IDS. In most systems any expansion of the signature-set being applied necessarily degrades system performance.

One way of understanding why the state-of-the art systems appear to have missed this important approach is related to the way in which misuse/signature-detection systems (and decision procedures in generally) are naturally generalized. It is easy to understand that “exploit polymorphism” and the high cost of monitoring an expanding set of threats immediately suggests the application of a statistical predictive approach. Signature-based systems quickly become unwieldy to maintain and costly to compute. As a result systems tend to either be bottom-up signature-based approaches (with low rates of false positives) or top-down anomaly detectors (with low rates of false-negatives) [56]. Our contribution is to show that in some cases detector performance can be incrementally improved using statistical predictors, building the *IDS* “toward the middle”. The basic idea seems sound. It also represents a natural progression towards more sophisticated and (hopefully) better detectors. If we are lucky, we might even end up with more robust and resilient detection.

A stronger statement concerning the purpose of statistical methods might also be made. Clearly, statistical approaches are poor at identifying the nature of particular events precisely. It might be claimed that the fundamental purpose of statistical anomaly detection should not be for the detection of anomalies but in the prioritization of decision procedures which are good at precise prediction (at the expense of performance and coverage).

2.2.1 Host versus Network Detection

The particular detection model representation is not particularly important to our thesis. Numerous representations have been used, as described in the literature (e.g. rules, scripts, decision trees, neural networks, statistical models, Petri nets, etc.)[92]. As noted by Yu and Tsai[92], a signature-based system is often easier to modify dynamically due to the relative

Chapter 2. Background

independence of individual signatures, although it is clear that performance specifics are dependent upon the underlying implementation.

Regardless of the detection model, IDS can be loosely grouped into host and network-based systems. Host-based systems can easily rely on limiting detection to those attacks that might possibly affect the software which is actually resident on the host. Network-based detections however, can only take such an approach if given access to data structures containing the status of host configurations and software. Limits will always exist in knowing the precise state of host machines unless a host-based sensor is used. However, host-based sensors are not always possible or even desired due to the possibility of host-compromises, the latency of centralized storage, and the potential for data corruption. Host-based and combined systems are not tolerant to corruption of host state. Such approaches also require significant resources to maintain shared state between host and network-based sensors. Although combined systems are deployed within many operational environments, engineering compromises and overall cost are often prohibitive, or result in unacceptable compromises.

Network-based detections are required to detect many classes of attack (e.g. distributed denial-of-service, some types of scans, spoofing techniques, etc.) Network-based detection is also required in order to guarantee that hosts which may be compromised still have a detection and reporting mechanism in place. Nonetheless, knowledge of machine state is desired and even necessary for making IDSs both tractable and accurate. Knowledge of the state of a potentially vulnerable host can elucidate the set of signatures that are relevant. Knowledge of the state of an attacker's machine can provide insight into the goals, exploits, and knowledge of the attacker. This knowledge can significantly affect the confidence in our knowledge of the state of an attack and guide both our detection strategies and our responses. Unfortunately, although host-based sensors have been widely used in the context of signature-based IDS, comprehensive victim and attacker modeling has not.

2.2.2 Misuse and Anomaly Detection

Anomaly detection system (often called “behavior-based”) utilize learned statistical distributions of normal network and user behavior in order to detect significant deviations. Systems can be trained to perform both the detection and classification of anomalies, but do not provide an obvious semantics to elucidate the meaning of an alert. This limits their direct use for understanding the meaning of detected events. Such systems rely on information provided by other threat and attack modeling techniques which is an area of particular relevance in extending existing IDS capabilities using predictive techniques[31]. Nonetheless, such systems offer generality that misuse detection does not, detecting malicious activities which a misuse detector has no patterns for.

Anomaly detection systems are by definition predictive. By using representational statistics to gauge the likelihood of an event, they are used to predict whether activities are benign or may represent a threat. In the simplest instances, a detected event which is determined to have a low likelihood in a simple Markov model and a fixed or dynamic threshold determines whether the event is anomalous. For example, one of the first of such systems, Time-based Inductive Machine (TIM), used a set of inductively learned temporal event sequences which were used to detect deviations from established activities and unrecognized event sequences as anomalies[87]. Systems such as TIM are absolutely essential to enable the detection of new threats, but do not directly address the issues of scalability for precise event detection.

This dissertation considers equivalent predictive models to those commonly employed in anomaly detection, but expressly for the purposes of increasing coverage and improving performance. For example, in the case of a Markov predictor, predicted future states in a learned transition matrix are not used to determine anomalies, but to prioritize the detector’s decision procedure. In essence, those events that are *more* anomalous are presented with a larger decision procedure, whereas events which are *expected* are presented with a smaller

Chapter 2. Background

decision procedure representative of the expected outcome. Other anomaly detectors generally apply these predictions directly for the purposes of detecting anomalies, not for the improvement of the detection mechanism itself.

Signature-based IDS are precise and due to their simplicity have been employed within many large-scale, commercially available systems. In such systems each input (e.g. header, packet, TCP session, event, event sequence, derived feature set, etc.) is compared against sets of thousands of signatures in a more-or-less brute-force manner. The patterns used and the algorithms employed for pattern matching are very efficient. Significant gains have been made in the last two decades in the area of pattern matching. Of particular relevance with respect to performance are a predictive matching algorithm developed by Vespa, et al. and the substantial previous work with content-addressable memory for pattern matching[90]. These classes of predictive approach are at a very fine granularity with respect to the semantics of network events. These techniques generally improve the performance of every pattern assessed by a system, leading to substantial performance improvements.

Commonly, signatures are written to be very precise with respect to vulnerabilities, exploits, and other “known bad” sequences. False negatives are common and false positives are often manually “tuned” out of signature-sets over time[35]. Such systems have the benefit of precision, but are generally poor at detecting new exploits and are very labor intensive to maintain in the face of large numbers of vulnerabilities and exploits. The signature-tuning process often consumes a significant portion of the time spent managing a signature-based system’s signature-set. A number of adaptive systems and automated signature generation system have been proposed and constructed to alleviate some of these issues[23, 24, 92]. In many instances adaptation occurs only in direct response to operator feedback in identifying false positives or operator information overload.

These systems are also often limited by particular nuances of the signature language used and the specific preprocessors which are employed. Further, there are classes of

attack, which due to dynamics in the patterns produced, are simply not detectable using conventional signature-based approaches[16].

2.2.3 Signature-Based Detection

The most common and well-understood misuse-detection approach within modern IDS is to utilize one or more pattern matching signatures for each identifiable attack or compromise. This is commonly termed “signature-based detection” and is often desirable because of its precision and simplicity.

A “signature” in IDS parlance is a set of patterns (or in some cases preconditions) which identifies a known threat (as uniquely as is possible or practical) [23, 82]. Signatures in many detection systems are often very precise and consequently expensive to compute. These systems have a resultingly low false positive rate, but generally require at least a single signature for each vulnerability, exploit, or attack vector (each signature representing a series of potentially costly computations.) These signatures must often be kept as simplistic and as compact as possible for purposes of performance and this results in brittleness. Nonetheless, signature-based detection is often desirable due to its simplicity. There is a direct and easily understood mechanism which relates the features of network traffic to identification of malicious events.

Each signature is composed of explicit patterns, bytes, strings, field values, expressions, and preconditions. Signatures are most often pre-filtered by matching against protocol attributes first (both IP packet and TCP session headers and payloads) and expensive string matches and regular expressions second. Signature-based approaches can be performed very efficiently using pre-compiled expressions, careful management of feature ordering, and automated binning and pre-filtering of incoming traffic such that signatures only “see” traffic that is relevant (e.g. by static fixed signature fields of protocol, port, address, etc.).

Chapter 2. Background

Unfortunately, signature-based intrusion detection system approaches seem to have been recently neglected by the research community. Few significant gains in the tractability of IDS have been made in the last few years while many of the scalability issues have been hidden by modern achievements in computing performance and parallelization. Many previously identified issues have simply been swept away by engineering slight-of-hand, current systems making gross compromises in the name of engineering feasibility and often requiring a highly trained staff to manage even a single deployed IDS[59].

Within existing threat detection and threat response systems, design and configuration compromises abound. In particular,

- Computationally short-circuiting the detector to only alert on the first (or a limited number) of detected events can allow attackers to “game” intrusion detection systems by flooding these systems with irrelevant information.
- Denial-of-service (DOS) and flooding attacks often cripple IDS and prevent detection of more important attacks masked by the DOS.
- Tuning an IDS to achieve better performance (by removing apparently irrelevant, old, or noisy signatures) decreases the IDS coverage of existing threats. The resulting poor coverage of a “tuned” IDS can also allow obscure and “obsolete” attacks to older systems to go completely unnoticed.
- Removing signatures based on the need to eliminate spuriously large numbers of false positives allows attackers to circumvent detection by making it appear that an alert is a false positive. It is generally trivial to generate traffic to match a detection signature. Using crafted packet traffic, an attacker can generate false positives in obnoxious numbers, resulting in the alert’s subsequent removal (tuning) by administrators.
- Lack of adaptive mechanisms within IDS can allow new attacks to proceed unhindered and undetected[59].

Chapter 2. Background

- Lack of industry-wide standards, incentives, and protocols for sharing information (intellectual property in the form of IDS signatures/patterns) results in poor coverage for all.

The sizes of signature-sets is often kept as small as possible to allow detection systems to perform under peak network loads. Consequently, these system suffer poor recall due to lack of sufficient coverage of exploits and variants. These limited signature-sets often even preclude comprehensive coverage of expected vulnerabilities and exploits. The Snort IDS, one of the most widely deployed, currently includes approximately 18,000 signatures. This limited signature-set covers only a small fraction of the known threats.

For similar reasons, signatures are written to cover vulnerabilities rather than exploits, resulting in a lack of insight into the active attackers methodology and toolset. Without these insights, defensive postures and response actions are generally performed blindly. It is as if we can recognize that the shooter is using some form of projectile weapon through an office window, but we have no idea whether he is using a rock, a .22, or a bazooka.

Even with severely limited signature-sets, the flood of resulting alerts is often too much for a human analyst to reason about, requiring complex aggregation systems to cluster, correlate, and corroborate a fire-hose of (mostly) irrelevant information. From an analyst's perspective the deluge of data produced by an IDS has the outward appearance of high-coverage, scalable detection. The essential approach to the dealing with the fire-hose is to filter, aggregate, and annotate known attacks and unknown anomalies[92]. In theory and practice this helps, but does not address (and actually serves to hide) the underlying problem of the poor coverage and poor performance of the detector. As a result, signature-based systems tend to be either *too brittle* or *too expensive*. And in a complementary manner, anomaly-based systems tend to be either *too noisy but robust* or *acceptably quiet but useless*.

The daunting scale and scope of future requirements is likely to make brute-force IDS intractable without a huge economic cost in hardware and manpower. Arguably, such

systems currently achieve only marginal utility, representing simultaneously (and non-intuitively) both the best precision and the poorest coverage of the actual attack vectors and related network activity.

2.3 Characteristics of Signature-Based Detection

2.3.1 Decision Procedure

There is no single model which adequately describes the wealth of approaches found within existing IDS products and research efforts. Nonetheless, for the purposes of discussion, we can treat most signature-based IDS as if they were simply some type of optimal decision tree. Indeed, the more complex nuances are often treated as special cases after a fast decision procedure has first been applied.

Decision tree techniques have a long history as key components of detection systems[8, 53, 60, 61, 73]. In order for host or network-based IDS to be tractable the detection scheme must be significantly more efficient than an exhaustive comparison of signatures to packets and packet-derived features. Decision trees provide one possible mechanism for improving IDS performance. Even systems with sophisticated pre-filtering and pre-processing mechanisms spend a significant amount of energy processing data-irrelevant signatures. Decision-tree approaches can improve performance both in achieving log-scaling in respect to signature-set size and by organizing the structure of trees using Shannon-entropy and other information theoretic methods as pioneered and elucidated by Quinlan et al.[53, 75, 76, 77].

Although a clear improvement over any type of brute-force signature-based system, there are a number of outstanding issues with decision-tree approaches. The construction of such trees is both expensive and somewhat opaque. In many cases the entire decision

Chapter 2. Background

tree must be recalculated for each signature addition. Although this needs to be done only once per addition to a signature-set, it does not allow fast incremental and modular signature-set additions. Algorithms such as ID3 have a sub-exponential complexity, but is still proportional to $O(n |v| |r|)$, where n is the number of examples, $|v|$ is the number of features, and $|r|$ is the number of internal nodes in the decision tree [77]. Since a substantial benefit of modern signature-based IDS is the ability to “turn off” subsets of the signature-set and to dynamically generate new signatures based on the output of anomaly detection, rebuilding the decision tree for each signature addition appears highly undesirable. The more dynamic the IDS (or the larger the IDS), the less appealing such a decision tree construction method becomes.

Nonetheless, these approaches have significant performance gains over naive IDS signature matching approaches. Naive IDS signature matching approaches do not allow for any significant parallelism and the most common approaches often suffer from signatures duplicating a significant amount of work when (as is often the case) many signatures share the same or similar preconditions[53]. In ad-hoc approaches to improving parallelism, a set of well-known common constraints are used at the top-level to partition the signatures (e.g. ports, protocols, and IP addressing). Because a naive ad-hoc approach does not consider Shannon-information, these approaches are generally much poorer in performance than information theoretic approaches[53]. The naive approach hand-picks constraints such as port and protocol to partition the signature-set. Information-theoretic approaches pick the most discriminating features and performs parallel evaluation of every relevant feature[53]. For the purposes of later discussion, the iterative information theoretic approaches to constructing decision tree can be considered to be “greedy” tree construction strategies.

Another consideration is that while greedy approaches do construct an optimal decision tree given the entire signature-set, such a tree is demonstrably suboptimal for some signature subsets[42, 53, 77]. There are both online and offline solutions to this problem. If the signature-set were first clustered and partitioned to group sets of semantically related

Chapter 2. Background

signatures, a forest of decision trees can be constructed, each of which may perform better, and never worse, than a globally constructed tree. That is, in a sufficiently diverse signature-set, the decision with the maximal information gain for the entire set of preconditions for the full signature-set is not the same as the optimal decision for a partitioned subset. Pre-clustering and partitioning of the signature-set prior to constructing a decision-tree should produce superior performance with an increased cost to construct the decision tree. This, of course, is only true if we know a priori an ordering of the decision trees that we should first traverse (a decision tree of decision trees), which is a principle tenant of this dissertation. If we have access to the right prior information we can perform prediction to choose relevant subsets of the global decision tree in priority order and gain significant performance. Without such prior knowledge and prediction, the partitioned tree will perform poorly in comparison to the globally optimal tree because each subtree will need assessed in arbitrary order.

Another approach by Friedman et al. is to construct a decision tree in a “lazy” fashion which constructs and caches an optimal decision tree for each test instance[42]. Their approach works primarily due to the lazy algorithm being provided with the additional information of the test instance (in addition to the training instances). The algorithm has $O(m n)$ complexity for m instances and n features but achieves better performance scaling due to caching of commonly used paths, giving better performance at a substantial memory overhead. The end goal of the “Lazy Decision Tree” approach is to achieve an average tree depth with is smaller than $\log(n)$. As will be shown in Chapt. 3, a simple model of an anticipatory approach can be understood in terms of decision trees where prediction of equivalence classes (and smaller sets of decision trees) achieves performance gains by the decrease in problem size. An approach such as “Lazy Decision Trees” could provide a straightforward mechanism for generating more compact decision trees and applied to the problem at hand.

2.3.2 Signature Information

An analysis of a recent Snort signature-set shows that a significant amount of effort must be expended to match against literal and regular expression content matches. With respect to string matching, over half of the 53,000 current signature content options provide a significant 4.725 bits of information. The distribution of information per signature is also bimodal, meaning that most signatures are either unique and not shared across multiple signatures, or are essentially identical. There is no obvious way to infer a total ordering of these signature comparisons in a decision tree. As a result, existing decision tree methods will order the signatures arbitrarily.

By default, many systems accomplish efficient signature matches by binning signatures and packets together using high-level features that are common across all signatures (e.g. port, protocol, network segment, ...) and only then applying costly string and regular expression pattern matching techniques. Systems may also shortcut further processing after the first signature resulting in an alert. If we want a truly robust IDS that does not depend upon signature ordering we must check every signature to find the most relevant or critical alert. If we had prior knowledge of attack intent, latent vulnerabilities, attack scripts, or other information we could use this information to order the application of signatures.

Systems such as Snort use engineering optimizations and context-dependent comparisons to make signature comparisons as fast as possible. However, the inability to structure content comparisons into any type of decision tree means that there is a significant disincentive to broadening the scope of IDS signatures to encompass a wider diversity of attacks and attack signatures. It is the purpose of this research to show how matching against an enormous set of extremely precise signatures can be made tractable by circularly using this precision to predict the currently relevant signatures for each connection. Adding additional signatures in this way adds precision and improves recall but should not significantly increase the cost of performing pattern matches.

2.3.3 Alert & Signature Semantics

The structure of most IDS is often very opaque. Most of the semantic preconditions for a signature firing (and causing an alert) are hidden within the architectural and engineering design of the IDS. Although signatures often appear simple, they may represent a fairly sophisticated grammar with additional preprocessors being capable of general computation [79]. If we desire extract a rudimentary axiomatic semantics of a single signature we might be able to use the signature-language, expressions, and configuration state of the IDS. However, a complete understanding and axiomatic logic specification of IDS signatures is problematic due to interdependencies in the interpretation of signature attributes and the hidden assumptions in the IDS architecture. Therefore, these and other architectural semantics remain hidden. Because the signature semantics of existing commercial IDS is essentially hidden, the preconditions cannot be efficiently extracted and used to create more efficient decision procedures which isolates small sets of signatures that apply to the particular datum in question. Temporal pre- and post-conditions are generally undefined and guarantees on computing costs for a particular signature are unknown except in the simplest cases of literal byte and pattern matching (related to the well-known halting problem).

The semantics of an IDS signature can be stated explicitly as a set of preconditions. These preconditions can then be used as annotations and a hierarchical clustering algorithm can be run to construct a dendrogram of preconditions. Each node in the tree represents one or more potential events as post conditions. Note that in such a tree, a conventional IDS is essentially all of the leaves of this tree. A conventional IDS performs a matching of each packet with each leaf. A more efficient approach would be to distribute the precondition semantics throughout the tree. General-purpose preconditions would be assessed first and thus prune the tree in a top-down manner.

Rules within most IDS are best described as a context-sensitive grammar which uses embedded pattern matchers in a hierarchical decision-tree matching algorithm. These

Chapter 2. Background

grammars are context sensitive across the tested features, but do not include temporal aspects (although many preprocessors include analysis of temporal features). The Snort IDS has numerous signatures defined for a large number of software vulnerabilities. If we start by looking at the simplest class of signatures (those for Internet Control Message Protocol (ICMP) packets) we can show how these simple cases can be transcribed into a formal logic and then extended for the purposes of precision or generalization using temporal logic. Examining a standard signature from the “icmp.rule” file from Snort release 2.9.1.2:

```
alert icmp $EXTERNAL_NET any -> $HOME_NET any
  (msg:"ICMP ISS Pinger";
  itype:8;
  content:"ISSPNGRQ"; depth:32;
  reference:arachnids,158;
  classtype:attempted_recon;
  sid:465;
  rev:4;)
```

Listing 2.1: Example of a Snort Signature

This signature is representative of a large number of signatures within the Snort signature database. The port, protocol, and address features serve an important role in partitioning the incoming network traffic into high-level bins. Packets never see signatures which reside within bins for different IP and TCP header features, which are inexpensive to check using integer comparisons. This signature can be written directly in a predicate logic as:

$$\text{proto}(P) \wedge \text{src}(E) \wedge \text{dest}(H) \wedge \text{icmpType}(8) \wedge \text{contentMatch}(\text{"ISSPNGRQ"}, 32) \rightarrow \\ \text{reference}(\text{arachnids}, 158) \wedge \text{classtype}(\text{attempted_recon}) \wedge \text{sid}(465) \wedge \text{rev}(4)$$

where $P := \text{ICMP}$, $E \in \text{EXTERNAL}$, and $H \in \text{HOME}$

Listing 2.2: Predicate logic representation of a Snort Signature

This attribute-value semantics makes no consideration of multiple ICMP requests or other attributes that are not present within the packet payload. It also makes no determination

Chapter 2. Background

of how the packet was constructed, contextual aspects such as timing between this alerted packet and other traffic, where the packet originated, etc. We can add a temporal relation to our semantic description to extend the original semantics directly. If using solely the individual packet characteristics we can specify relations which encompass expectations about attacker or scanner behavior. In the above example we may use:

$$\text{alert}(465) \wedge F(\text{alert}(465.5)) \rightarrow \text{exists}(\text{sourceAddress})$$

Where the eventually operator F is defined in Table 2.1. This means that if eventually we see an alert with a Snort Identifier (SID) of 465 (indicating detection of some type of attack) and at any future time see alert with SID 456.5 (indicating host response to the attack) then we can assert that the host with *sourceAddress* exposed its existence. The SID was contrived for this example, but represents the addition of a signature to alert on exposure of information about hosts which are the victims of reconnaissance or attack. Of course, exposure of a host is not particularly important except when in the context of an active attack. Gating collection of secondary information such as host exposures may be a reasonable method of limiting superfluous event output. Figures 2.4 and 2.5 illustrate the Next and Finally semantics over packets and alerts.

Table 2.1: Linear temporal logic relations

Symbol	English	Meaning
$N\phi$	next	ϕ must hold at the next state
$G\phi$	always	ϕ must hold on the entire subsequent path
$F\phi$	eventually/finally	ϕ eventually has to hold (somewhere on the subsequent path).
$\psi U \phi$	until	ψ has to hold at least until ϕ , which holds at the current or a future position
$\psi R \phi$	release	ϕ has to be true until and including the point where ψ first becomes true; if ψ never becomes true, ϕ must remain true forever.

There are also no asserted post conditions concerning the fact that this packet was likely generated by the Internet Security Scanner (ISS) in order to gather information about hosts. The IDS signature is designed to detect, but does not itself help identify any of the interesting information regarding the exploit being used, the likely intent of the attacker, or host exposure information. Use of temporal logic can assist in detecting some forms of attack, but more importantly it helps identify other information that is potentially useful in

Chapter 2. Background

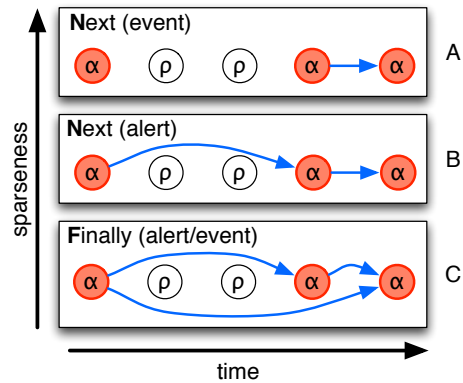


Figure 2.4: Temporal logic relations over IDS events

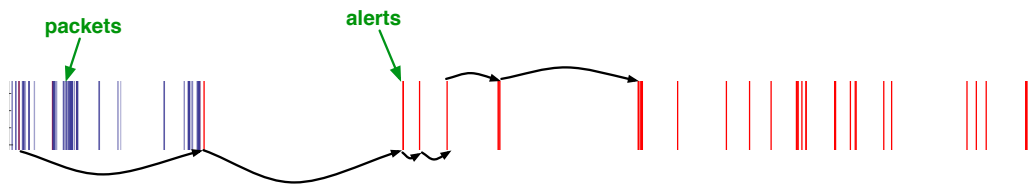


Figure 2.5: Temporal logic using Next(alert) semantics

modeling attacker intent and identifying attacker knowledge which directly results from host exposure. In the case of an ICMP scan we might increase a measure of threat severity by whether the threat is widespread or targeted.

2.3.4 Traffic Patterns

An obvious way of defining patterns within network traffic is based on the IP or TCP header. We can define a connection as being, a 5-tuple corresponding to an active TCP connection, as the 3-tuple consisting of only a pair of IP addresses and an IP protocol. The way that we define a “connection” can affect the distribution of learned patterns. TCP connections are not always long-lived, many only lasting seconds and consisting of only a handful of packets. IP connections may learn nonsensical associations due to network address

Chapter 2. Background

translation (NAT) or proxy configurations. Neither short-lived TCP connections, or broadly defined IP connections will provide a sufficient basis for anticipation in all cases.

Nonetheless, TCP connections provide a useful basis for collecting sequence statistics. The general class of traffic for future TCP sessions is very likely to be the same as prior sessions on the same server port. For example, a client which is retrieving a web-page will result in multiple TCP connections being formed for accessing various data elements of the web page, some of which may even reside on different physical machines. Future TCP sessions for the same pair of IPs are likely to contain Hypertext Transfer Protocol (HTTP) data. An anomaly detection system might use this to detect and alert on deviations. An anticipatory system can bias a signature-based detector to focus on threats relevant to serving and retrieving web-pages.

Similarly, IP connections can usefully expose information about to whom individual machines are likely to communicate. If two machines communicate regularly the fact they they communicated is less interesting than two machines which communicate infrequently. Capturing this information and statistics about the type of communication can assist in determining which connections are interesting for purposes of sequence prediction. Alternatively, connections through Network Address Translation (NAT) or a proxy would appear to be somewhat uniformly distributed. Deviations from this distribution may be just as useful.

2.4 Per-Signature Cost Optimization

Often, a single detection signature can account for a significant portion of the processing costs for an intrusion detection system (IDS), particularly in the cases when an ongoing attack is present, when a signature has a high computing cost innately, or when a signature is active for a large portion of the input data-stream. In each of these cases, there are existing engineering optimizations which either decrease the computing cost (by eliminating or

Chapter 2. Background

chaining signatures) or decrease the number of events produced (by eliminating events entirely, limiting the number of occurrences, or limiting the number of duplicates).

Suppressing output events generally has very little effect on system performance. The approach is often used as an aggregation method or to remove event output which is unimportant but cannot be easily disabled. Alternatively, both signature-chaining and signature-removal can improve performance, but provide relatively narrow improvement. Signature removal guarantees false negatives for all events associated with the removed signatures and therefore is not appropriate in all cases. Signature chaining is only applicable for events for which prior packets can be used to prevent signature activation. We show that there is another important class of optimization which can further improve performance, particularly when a system is under load and dropping input packet data.

2.4.1 Flow Tracking

One class of optimization which is commonly incorporated into modern IDS is flow tracking. This is a mechanism for tracking the state of individual connections such as the state of a TCP connection: whether it is established, its direction (client initiated or server initiated), and whether the connection has been closed and by whom. A system which did not track session state would likely require more complex signatures, be incapable of detecting some types of events without additional false-positives, and spend significantly more time processing signatures that would not have been processed had the session state been known.

2.4.2 Signature Chaining

Signature chaining is closely related to and often dependent on flow tracking. The mechanism as employed within the Snort IDS is called “Flowbits” and allows a signature to set a bit flag which is subsequently checked by other signatures. If the status of the flag

Chapter 2. Background

meets the signature's requirements (being either positively or negatively conditional on the bit flag being set), then the signature is activated. Any number of Flowbits can be set or checked by a set of signature resulting in fairly sophisticated signature dependencies. Of primary interest in respect to this paper is the use of Flowbits to keep high-cost signatures disabled until they are relevant. This decreases the costs associated with signatures which may otherwise overwhelm resources available by being applied to the wrong packets.

One limitation of the existing signature chaining mechanism within Snort is that no guarantees are made regarding setting or checking of flowbits for a single packet between multiple signatures. Depending upon which order the signatures are processed in, setting a Flowbit in one signature may or may not activate signatures which are dependent upon the bit being set. This has implications for the thesis of this paper as we make use of the Flowbits mechanism to probabilistically activate signatures.

Another issue with signature chaining mechanisms occurs when there are negative conditional dependencies. If the firing of a signature that sets a Flowbit fails to occur, then a signature which is negatively dependent on that Flowbit will be left active and may result in an increase in false negatives. In respect to our overall approach, this is the only known cause of additional false negatives above the control configurations.

2.4.3 Event Filtering & Suppression

Event filtering is often used as a method of decreased the total number of events which are output. Since many detection systems operate over individual packets, some events will occur as frequently as the incoming packet stream containing the relevant set of detected features. Large numbers of duplicate events does not provide much additional information (only that an attack is still ongoing) and may overwhelm an operator or an external analysis system. Event filtering and suppression, however, do not generally save computing time related to the costs of processing incoming packet data. Even though an event may be

Chapter 2. Background

configured to be output only once each second for a given IP connection, many such events may have been computed by the IDS. Each event which is filtered represents wasted effort by the IDS.

Fortunately, it is fairly straightforward to decrease the number of times that a signature is used. To this end, we have defined a set of signature-activation policies to explore the effects of various signature activation functions. In the next section we then provide a detailed analysis of false-negatives given certain assumptions concerning packet loss. To support our claims we describe a prototype system and experimental results. The prototype takes direct advantage of the existing flow-tracking and signature-chaining mechanisms within the widely available Snort IDS and provides significant gains when overburdened.

2.5 Related Work

There are many approaches to detecting network-based computer attacks and numerous surveys of their history, development, benefits, limitations, and recent achievements.

The basic principles of computer intrusion detection were first thoroughly described by Dorothy Denning in her paper “An Intrusion-Detection Model”[31]. Denning’s paper provides a foundational perspective on IDS which particularly helpful in understanding specific advancements in IDS in the last 20 years. All IDS can be understood in the context of her model, irrespective of how an IDS is deployed, the type of detection mechanism used, or how IDS output is used. Her model was implemented in the Intrusion Detection Expert System (IDES) developed for SRI International in the late 1980’s[66]. Denning’s model describes subjects, object, audit records, profiles, anomaly records, and activity rules. Audit records describe actions between subjects and objects. Profiles characterize the “normal” behavior of a given subject in respect to sets of objects. Anomaly records are

Chapter 2. Background

generated by an IDS when an activity rule's preconditions are met. Activity rules link a set of preconditions to an action to be taken.

Denning's model also describes a system which potentially spans all facets of computer software, hardware, and networks, monitoring and alerting on activities which span software instances, computer hosts, and network segments. However, few existing systems are so comprehensive, their innate complexity and maintenance costs being prohibitive.

While many detection systems have existed solely for the purposes of research demonstration, there are dozens which have had widespread use. Systems such as Snort[79], Bro[72], and Suricata[4, 5, 28] have become readily available as open-source. This has allowed many independent researchers to explore their function and limitations. The Snort IDS, has substantial use within operational settings and is the de facto standard for comparisons between approaches.

However, of particular interest are systems which attempt to achieve high coverage of threats and relevant characteristics. My personal introduction to intrusion detection systems began in late 2002 while working for the Defense Advanced Research Project Agency () with a system which was inspired by Denning's general-purpose model[43]. The system, developed by Australia's Defense Science Technology Organization, known as *Shapes Vector*, encompassed many of Denning's concepts[9, 10, 11, 12, 13, 37]. It utilized a collection of agent-based expert systems to learn facts about the network and network activities. Individual agents performed protocol analysis, correlated activities discovered from direct monitoring of the network, monitored of host and network audit logs, monitored of the output of independent intrusion detection systems, and performed both forward-chaining (data-driven) and backward-chaining (goal-driven) deduction.

The key characteristic of this system was that its expert system components were used as unbiased observers, asserting facts deduced from incoming data irregardless of whether these facts were part of an active intrusion. This system used the output of systems such as

Chapter 2. Background

Snort as individual agent inputs. In other words, the coverage of this system was essentially exhaustive, covering identifiable threats (by multiple independent IDS) as well as all of the host, network, and transmission characteristics known to the agent expert systems. As later chapters will discuss, it is the coverage of an IDS which primarily determines whether anticipatory approaches can be used for performance optimization.

In respect to Denning's original model, this system is the closest which I have personally seen in terms of its comprehensive coverage and general-purpose intent. As in many other systems, however, detection is performed in a purely feed-forward manner and speculative optimization of detection engine performance is sorely missing.

While the "Shapes Vector" system's potential capabilities were powerful, the need for hardware was directly proportional to the problem size and the amount of incoming data. Needless to say, the high cost of the system both in hardware and administration have made it difficult to apply to real-world environments. Hardware is almost always wanting and domain experts who are able to modify and maintain such a system are few and far between.

It is my humble perspective that it is exactly Denning's original vision (and exemplary implementations of this vision such as the Shapes Vector systems and others) which would most readily benefit from anticipatory optimization approaches.

2.5.1 Attacker Modeling

One facet from commonly missing from available IDS is any deep knowledge of attacker intent. In respect to our research goals, knowledge of such intent would be very helpful for the prediction of future events.

The most common method of modeling attacker intent is through the use of attack graphs. Each path through an attack graph represents a possible set of steps that an attack might take to achieve a given goal. Correlating events which occur at different times or at

Chapter 2. Background

different points on a network is generally difficult. Recent work by Roschke et al. shows how alert correlation can be performed efficiently using attack graphs and a matching function[80].

In 2008 Zhang et al. extended the concept of attack graphs and describe an *attack grammar*[93]. Their primary goals were to provide a more compact representation which could leverage straightforward syntax checking for the elimination of detector errors. While equally powerful to attack graphs, an attack grammar also has the benefit of being more easily represented (and understood) visually. Similar to attack graphs, the principal question being asked is related to the well known graph theory problem of *reachability*. The new representation can also be easily converted from various attack graph formats. Although there is still no inexpensive way of generating the grammar itself, the intent is that once the grammar is generated it might applied in many different scenarios before new grammar rules are needed.

In respect to the generation of attacker models there have been several significant recent research efforts. In particular, Zhu and Ghorbani describe a new approach to generating knowledge of attack strategies using multilayer perceptron neural networks and support vector machines[94]. Their “Alert Correlation Matrix” is closely related to the Naive Bayes method described in Chapt. 5. Unfortunately, their use of the 1998 training set means that some of their results may be peculiar to the dataset (see Sect. A.2.1 on page 160). The attacks, attack sequences, and background traffic present within the Defense Advanced Research Projects Agency (DARPA) dataset are not distributionally similar to real-world data. Nonetheless, given sufficient data, automatic generation of attack graphs using their method seems feasible.

2.5.2 Prediction & Performance Adaptation

There are a number of issues with signature based IDS as they are commonly implemented. Of primary importance is the conflict between coverage and performance (i.e. precision, recall, accuracy, and specificity). Some form of cost analysis (e.g. computing, latency, hardware, training, etc.) is generally performed in order to choose the right IDS technology for a given network. Many modern IDS also rely on labor-intensive tuning and signature-refinement to match particular network characteristics and known host vulnerabilities. There are many trade-offs which result in sub-optimal detection, but which decrease the IDS cost substantially. Fan, et al. describe cost metrics in terms of operational costs, attacker induced damage, and incident response, citing the need to consider cost within the development and deployment of IDS[39]. Others have cleverly incorporated cost assessment into decision support systems to propose or enact response actions [86], IDS reconfiguration, and dynamic performance tuning [58].

A commonly cited statement in the literature is that the number of alerts generated by modern IDS is overwhelming to human operators and essentially untenable for any type of manual analysis [58, 92]. Any number of false positives increases the burden without improving the abilities of the analyst. This statement is often cited as a failing of anomaly detection schemes and a justification for the elimination of “unimportant” signatures from misuse detection systems. It has also spurred the industry into the creation of Security Incident Event Management Systems (SIEM) which aggregate, correlate, predict, and display events and their impact[58]. While necessary for the sanity of the human analyst, a SIEM system does not address the issue of improving IDS coverage and can mask performance issues in underlying detection signatures. Due to the unresolved human-factors and cognitive issues, the research community has been essentially unmotivated to significantly expand the coverage of IDS. The number of signatures has remained relatively stagnant for nearly a decade. If the aggregation and correlation issues are ever adequately

Chapter 2. Background

resolved, the broader research community may return to identifying a wider variety of events using signature-based approaches.

Careful management of signature-set in order to achieve desired performance goals may also mask performance issues and can actually decrease the usefulness of IDS by removing contextual knowns from the stream of true-positives being displayed to an analyst. In a perfectly “tuned” system one might expect only the most high-priority events to be displayed and all other events to be discarded (or at least hidden). It is possible that removal of true-positives up front in the detector is only necessary due to the fact that such systems are not using predictive mechanisms to “bootstrap” their own performance. It is important to note that the predictive adaptations employed in many prior research studies have generally applied globally to the input data rather than being applied to individual signatures or to sets of signatures grouped by equivalence classes of *future* events.

A 2008 study by Yu, et al. demonstrated a system which tunes the detection model on-the-fly according to feedback from the system operator when false predictions are made. This adaptive anomaly detection approach throttles alarm output and tunes the detection model[92]. As such, these predictions are not directly seen by or affect the detection model.

Another way to think about tuning procedures such as those demonstrated by Yu, et al. is that they assist the detector in eliminating false positives. But for any set of signatures for which the false positive rate is already zero, no additional gains can be achieved. The procedure does not improve the performance of a signature-set other than to increase the accuracy. If false negatives are being incurred due to packet loss, the approach as described would not directly apply. Nonetheless, these results are important and demonstrate important methods for improving IDS performance. The general approach could also be easily modified to adapt the detection model to decrease packet loss and thus false negatives.

Chapter 2. Background

Another type of performance adaptation is one proposed and implemented by Wenke Lee & Wei Fan, et al. They describe methods for performing cost-sensitive adaptation of detection models[39, 56, 58]. Their primary contribution is to show that cost-based adaptive reconfiguration is a viable approach for performance optimization. The primary cost measures considered within their approach were: *taxonomic prioritization based on event type*, *damage cost*, *response cost*, and *operational cost*. Fan and Lee et al. desire to construct a general purposes cost-model and produce rough categorizations of each measure for purposes of system evaluation. Their basic approach is to reconfigure the IDS as a whole based on in situ measurement of performance issues. Particularly clever is Lee's use of injected events to determine when the IDS is dropping events. Lee proposes a set of cost objective functions for evaluating and optimizing detection performance[56]. Our research efforts serves to greatly expand their *operational cost* measure by exploring concepts such as wasted information and anticipatory performance optimizations.

Lee's approach is quite general and captures the fundamentals of performance adaptation and optimization. By formalizing each of the cost factors involved, Lee has created a relatively straightforward value optimization problem. This provides a useful global optimization for cases where the IDS is overloaded. However, during normal running of the IDS, in respect anticipatory and probabilistic refinements alike, the IDS is wasting work for events which are easy to predict or which are likely to occur many times during a single packet stream.

Similar to the probabilistic signature activation approach, Lee's threshold-based re-configuration is performed at the last possible time, when the system is failing. Such an optimization can only used to improve the IDS coverage during periods of high load. Further, the optimization problem is equivalent to the Knapsack problem, which is NP-complete and difficult to recompute online. Depending upon the number of features being considered, the cost of performing the optimization could be quite high and could not be used for online optimization. Although, online optimization for the lifetime of the IDS

Chapter 2. Background

instantiation is not considered in his analysis, it is clear that an extension might allow for parametric optimizations based on current event and detection engine statistics.

The anticipatory approaches outlined in this dissertation differ significantly from Lee. Such optimizations can be performed at any time, irrespective of system load. Within Lee's approach the optimization is delayed until the system is beginning to lose fidelity. It is also performed as an adaptation over the entire input set globally, which is likely to be sub-optimal for subsets of packet data. Instead, we consider an approach in which detections for each individual connection (or sets of related connections) are individually considered and improved.

In order for anticipatory approaches to be useful, IDS decision procedures must be constantly re-evaluating the most likely subsets of the decision procedure for all individual connections. Lee's approach adds a more comprehensive weighting regarding the relative importance of various signatures, producing an intelligent prioritization of event output, but this is optimal only *on average*. This is essentially the same problem inherent to global optimization of decision tree-based decision procedures. Lee is attempting to guarantee (based on numerous factors) that all events above a given threshold of importance will always be processed. When a performance threshold is met where the IDS is losing information, the system can be dynamically reconfigured to prioritize input data processing features by eliminating analysis tasks.

In the context of an anticipatory approach, Lee provides useful global optimizations, especially for the cases where the IDS is overloaded. However, during normal running of the IDS, Lee's approach is still wasting information gained due to processing of signatures for particular packets for which predictors might indicate irrelevance.

In addition to useful optimization schemes, Lee et al. also provide a wealth of arguments for the necessity of adaptive systems, but their goals are significantly different. Lee intends to ensure that when the IDS is overloaded optimal decision are made on what to keep and

Chapter 2. Background

what to throw away. He assumes that the analysis tasks already represent the maximum coverage possible.

Related to the work by Lee & Fan et al. is a 2003 paper by Balepin et al. which presents a single host-based detection and response paradigm. Their key contribution is a more comprehensive response cost model. The problems that they intend to solve are: a) ensuring that responses do not cause more harm than good; and b) ensuring that responses are not launched unnecessarily due to false-positives or contraindicating factors[17]. By taking into consideration potential response actions, they have provided a useful generalization from the cost models presented by earlier researchers.

When we discuss anticipation within the context of IDS what we are really after is anticipatory response. While our primary research goals have focused on anticipatory optimizations, the research community has been principally focused on the ability of systems to appropriately respond to detected threats. There has long been controversy in the design and deployment of automated response mechanisms, particularly when human decision making is removed from the loop. Nonetheless, it has become apparent that automated response is necessary and many systems incorporate automated response mechanisms. The simplest and most common types are those which make minor adjustments to prevent future attacks. These often adjust or stop network traffic flows (by dynamically changing firewall rules or dropping connections). Other systems may dynamically adjust security policies such as adjusting security domains within the software environment of a single host[85].

In 2009 Strasburg et al. refined the cost-sensitive detection concepts to better describe response systems[86]. They considered three factors: *response operational cost*, *response goodness*, and *response impact on the system*. While some factors used in their cost assessment methodology were subjective, even a subjective measure of cost can inform whether or not *any* response should be taken. If the cost of a response outweighs its benefits, then alternatives must be sought or planned responses abandoned. Cost-driven optimizations

Chapter 2. Background

such as those by Strasburg et al. can both enable better decision making by automated system and simultaneously allow more efficient use of limited computing resources.

More recent work by Barlet-Ros et al., while not directly related to intrusion detection, is relevant due to their use of predictive approaches for managing limited resources of network monitoring systems[18]. Their goal is to maintain bounds on network monitoring system accuracy by proactively shedding excess load. Similar to the PacketWrangler approach, they treat the monitoring software as a black box. However, their purpose is dynamic load shedding whereas PacketWrangler was intended for performance optimization even when systems are not overburdened. Their approach is novel, however, in that prior knowledge of cost models is not necessary.

2.5.3 Historical Surveys

For a broader perspective on intrusion detection in general, it is useful to review the descriptions and analysis of others who have reviewed the field. An exhaustive list of IDS is difficult to compile. However, most notable IDS have been surveyed by others in the field. Early surveys such as those by Snapp et al.[84] tended to focus on the basic principles internal to each IDS. Later surveys often ignore internal nuances and focus on system capabilities. The following lists notable IDSs seen in surveys over the last 2 decades. The year given is the year of the earliest cited paper within the surveys which describe the system.

In 1994 Jeremy Frank of University of California, Davis provided one of the first comprehensive overviews of the state of IDS technology after Denning's initial report[41]. His report is particularly interesting in that it focuses on the application of Artificial Intelligence (AI) techniques for intrusion detection rather than engineering aspects, focusing on Artificial Intelligence (AI) solutions to the challenges of *data reduction* and *classification*. He describes prior work using expert systems, rule based induction, classifier systems,

Chapter 2. Background

Year	System	Survey
1986	Discovery	[84]
1988	Haystack	[84, 41]
1988	Autoclass	[54]
1988	Multics Intrusion Detection and Alerting System (MIDAS)	[84]
1988	Expert System for Security Auditing (AudES)	[41]
1989	Wisdom&Sense	[84, 41]
1989	Information Security Officer's Assistant (ISOA)	[84]
1989	Computer Security Monitor (CSM)	[84]
1990	ComputerWatch	[84]
1990	Network Security Monitor (NSM)	[84, 41]
1990	Intrusion Detection Expert System (IDES)	[84, 41, 26, 66]
1990	Time-based Inductive Learning (TIM)	[41, 54]
1991	Distributed Intrusion Detection System (DIDS)	[84, 41, 26]
1991	Network Anomaly Detection and Intrusion Reporter (NADIR)	[41]
1991	Pattern Recognition to Anomaly Detection (PRAD)	[41]
1992	ASAX	[15]
1993	State Transition Analysis Tool (STAT) and USTAT (Unix STAT)	[26, 54]
1994	Tripwire	[26]
1994	Distributed program execution monitoring (DPEM)	[15]
1995	Next-Generation Intrusion Detection Expert System (NIDES)	[26, 54]
1996	Cooperating Security Managers (CSM)	[26]
1996	Graph-Based Intrusion Detection System (GrIDS)	[26]
1996	Janus	[15]
1997	JiNao	[15]
1997	EMERALD	[15]
1997	Bro	[15]

Table 2.2: Intrusion Detection Systems

neural networks, and decision trees. Frank also describes performance of several classifier algorithms in common use (Beam Search, Backward Sequential Search, and Random Search).

A dissertation by Kumar in 1995 also provides a fairly thorough overview of a handful of the systems developed within the previous decade[54]. He provides a thorough description NIDES, TIM, and STAT, providing enough detail to understand the guiding principles of their designs. In particular, he describes and reviews statistical approaches (NIDES), features selection approaches, predictive pattern generation (TIM), Neural Networks, Bayesian Belief Networks, Bayesian classification (Autoclass), covariance matrices (NIDES), conditional probability, keystroke monitoring, Expert Systems, state transition analysis (STAT, USTAT), model-based detection, and general-purpose approaches such as Denning's original model.

Chapter 2. Background

In 1996 Cannady and Harrel provided an accessible look into available intrusion detection approaches[26]. At the time IDS taxonomies focused on the differences between passive (prevention, preemption, and deterrence) and active (deflection, detection, and countermeasures) approaches.

A 1999 and revised 2000 report by Debar, Dacier, and Wespi of IBM Research Zurich describes a number of advancements and thoughts regarding IDS taxonomies[29, 30]. Their report focused on the efficiency of IDS in terms of measures of: accuracy, performance, completeness, fault tolerance, and timeliness. The taxonomy described IDS in terms of detection method, behavior on detection, audit source, detection paradigm (state vs transition based), and usage frequency. The approaches discussed by Debar et al. primarily focused on: expert systems, Petri Nets, statistical profiling, Neural Networks, and the Computer Immunology (particularly the work by Forrest et al. at the University of New Mexico[40]). Including those systems cited by Cannady, they compare over two dozen different systems in the context of their taxonomic classifications.

In 2000, Julie Allen et al. at Carnegie Mellon provided a thorough overview of IDS technology (under contract with Air Force Research Laboratory). Their survey identifies the principal characteristics of various IDS approaches, technology gaps, market surveys, subject matter expert surveys, policy surveys, and both organizational and technical recommendations[6]. Their survey is focused on commercial technology rather than research tools, but provides brief overviews of dozens of different detection systems.

Independent surveys in 1999 and 2000 by Axelsson[14, 15] presented a survey and taxonomy of IDS to date. Axelsson's taxonomy is both thorough and accessible. He also presents a review of several systems not mentioned in previous surveys.

During the same year, LaPadula of MITRE Corporation compiled an independent market survey listing many of the current anomaly detection and response tools with some minimal overlap to the Allen report[55]. Both reports are primarily descriptive rather than

Chapter 2. Background

exploratory. While only providing superficial detail their reports are the most comprehensive to date. Many of the technology gaps identified in the Allen report remain outstanding challenges today (performance issues, scalability, maintenance, growing attack diversity and attacker sophistication, detection of new threats).

A 2002 survey by Lundin and Jonsson[65] provides a more recent overview of IDS research, loosely describing current focus areas of: foundations, social aspects, operational aspects, testing and evaluation, IDS security, IDS environment and architecture, response, detection methods, and data collection. Their article provides a thorough description of various IDS and intrusion taxonomies.

The decade old articles by Kemmerer[51] and Verwoerd[89] in 2002 provided brief overviews of IDS approaches and problems. While many specific advances are obvious from bibliographic comparisons to earlier survey papers, the primary approaches and basic principals had remained the same.

Kabiri and Ghorbani presented a fairly comprehensive survey of IDS approaches in 2005[49]. They focus on network intrusion detection and some of the outstanding issues and modern techniques. They review prior work on Bayesian approaches, Fuzzy Logic, Data Mining, Genetic Algorithms, Rule-based expert systems, and specification-based approaches such as network modeling. They also spend some time reviewing the use of Honey Pots for identifying and delaying attacks in isolated environments prior to attacks on real systems.

A 2007 review by Stakhanova et al.[85] provides an excellent overview of various intrusion response systems and their characteristics. Their overview suggests that response systems are most usefully characterized by: autonomy, proactive/predictive power, adaptability, and cost-sensitivity. The characteristics most germane to this dissertation are those which are proactive or which consider cost-models in response planning.

Chapter 2. Background

More recent surveys have either focused on application-specific techniques (such as web-application security, or mobile ad-hoc networks) or come in the form of books. For a more academic overview of IDS techniques and technologies see: Bishop[19]. Additionally, many relevant publications appear in the proceedings of the Recent Advances in Intrusion Detection (recently renamed to Research in Attacks, Intrusions, and Defenses).³

³<http://www.springer.com/computer/security+and+cryptology/book/978-3-642-33337-8> - Research in Attacks, Intrusions, and Defenses, Springer Verlag (Website - Retrieved Sept. 22, 2012).

Chapter 3

Theory

In this chapter we explore a general model for anticipation within signature-based IDS. We first define the model and then support the need for such a model by describing how traditional IDS approaches waste information. We then define some of the relevant characteristics such as: *threat coverage*, *packet coverage*, *equivalence classes*, and present IDS *cost function*. In light of these characteristics, we explore potential performance gains given various performance parameters. We then look at a set of alternate (and supporting approaches) based on *probabilistic signature activation*. We perform an analysis of false negatives and support the claim that this type of augmentation is necessary for efficient IDS but is generally absent from existing systems. We balance the optimistic assessments by describing some of the errors which an anticipatory approach can introduce and discuss the trade-offs between IDS efficiency and error rates. We close the chapter by describing how the models presented lend themselves to game-theoretic approaches and suggest that Game Theory should have a prominent place in future detection systems.

The models presented are intended to stand largely on their own. Although many aspects of anticipatory detection are described in the literature, there does not currently exist a comprehensive model for anticipation mechanisms within detection systems. Recent

work by Barlet-Ros et al.[18] (predictive load shedding) as well as Hellerstein et al.[46] (predicting threshold violations) provide some basis for an anticipatory model but are limited in scope. There are also several models for IDS adaptation and reconfiguration based on system loads and attack prediction, but their applicability to the current discussion is somewhat limited (see Sect. 2.5 for a description of work by Lee et al.[56, 57] and Yu et al.[92]). There are also many models within and used to describe biological systems, but few are particularly relevant to the discrete event detection problem of network-based intrusion detection systems.

3.1 An Anticipatory IDS Model

One method to describe an anticipatory approach is to treat the IDS detection engine as a decision tree, and the predictor as a graphical model. At least for the forward IDS model, this is a gross oversimplification of existing systems. Nonetheless, it can provide a scaffolding for describing various aspects of an anticipatory approach.

A simplistic IDS might implement forward detection as a single optimal decision tree. This is consistent with the literature in which decision tree techniques have a long history as key components of detection systems [8, 53, 60, 61, 73] and allows for an easy analysis of an anticipatory approach. In the conventional model, this tree is pre-computed and provides a per-packet detection cost which is optimal on average. Figure 3.1 describes the architectural differences between conventional IDS and an anticipatory approach implemented external to the IDS (as in the Packet Wrangler prototype). In this model, a forward processing attack detector T_i is combined with an attack predictor G_i .

A conventional signature-based IDS (figure 3.1) simply map packets and other representations of network traffic into specific vulnerability and exploit identifications. In our approach, the detection engine is biased by prior events. Note that this diagram is

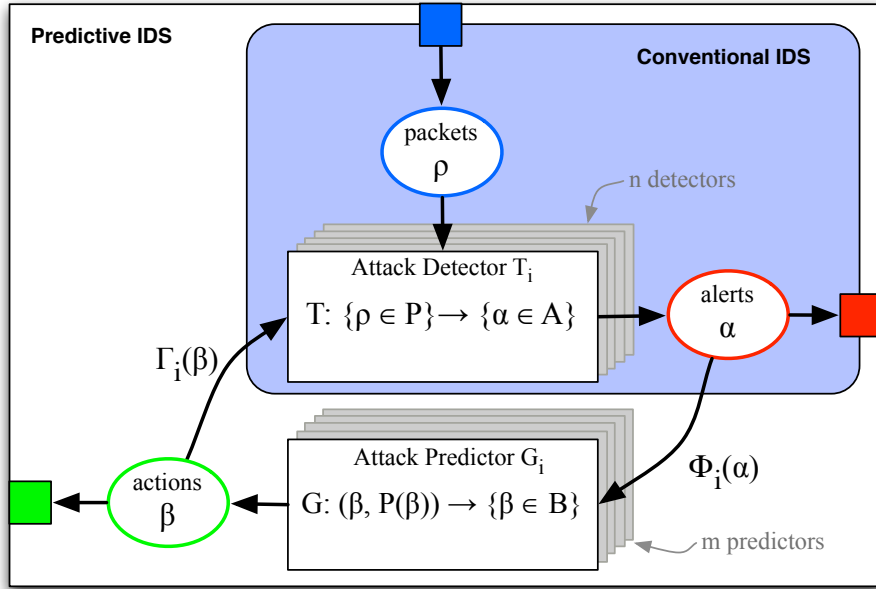


Figure 3.1: Predictive IDS architecture denoting conventional techniques

not intended to be all-inclusive of IDS techniques as many include sophisticated forms of prediction in the form of scripts (e.g. Bro IDS[72]) or entail threshold-based IDS reconfiguration based on evaluation of cost metrics and run-time performance (Lee et al.)[58]. It seems likely that each of these techniques might be improved using anticipation. An overview of the primary differences between our approach and other related approaches can be found in the IDS background chapter (Sect. 2.5 on page 40).

In an anticipatory approach, the global tree is paired with a forest of subtrees, T_{ξ_i} as in Fig. 3.2. Each member of the set of partitioned trees represents a set of alerts belonging to a particular attack equivalence class. These trees can be selected using the attack predictor (i.e. attack graph G_i) primed by prior alerts. For the sake of discussion, T is referred to as the *detection tree* and G as the *attack graph*, representing potential methods for detection and prediction respectively.

Using dual detection tree data structures requires at least twice the memory, but allows optimal cost traversal of either the global detection tree or a set of relevant subtrees de-

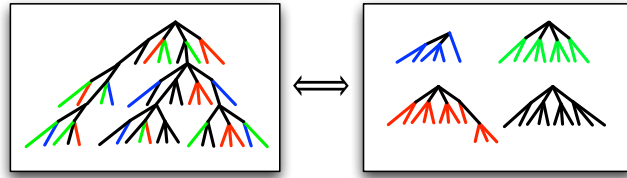


Figure 3.2: Dual decision tree representations with color-labeled equivalence classes

pending upon whether prior information is available to select a set of subtrees. A more clever data structure would be constructed to address memory issues if they were a limiting factor in a system's design. The average performance gain of the anticipatory approach is proportional to the average decrease in depth of traversal of the subtrees. This is almost certainly data dependent with some attacks being more prevalent, attack activity shifting over time, and each subtree potentially being of different sizes and structure.

In the model shown in Fig. 3.1, mapping functions $\Phi_i(\alpha)$ and $\Gamma_i(\beta)$ map alerts to predictor states and predictor states to equivalence classes respectively. For a given alert α , function Φ maps α into a set B_α of pairs of attack graph vertices and associated likelihoods, $(\beta_i, P(\beta_i)) \in B_\alpha$ (see Fig. 3.3). Γ maps attacker actions into alert equivalence classes ξ and associated probabilities $P(\xi)$.

The set B_α has a partial ordering based on the likelihoods generated by Φ . If subtrees T_ξ associated with possible future attacker actions are traversed in likelihood order, it should be possible to achieve a performance gain while simultaneously providing a prioritization mechanism. While signature prioritization is not always relevant, if limited computing resources are available due to high loads, eliminating unlikely subsets is more desirable than arbitrarily dropping packet data at the front-end of a system. Subtrees may be traversed in likelihood order until the set is exhausted, some minimum threshold criteria is met, or limited computing resources are consumed.

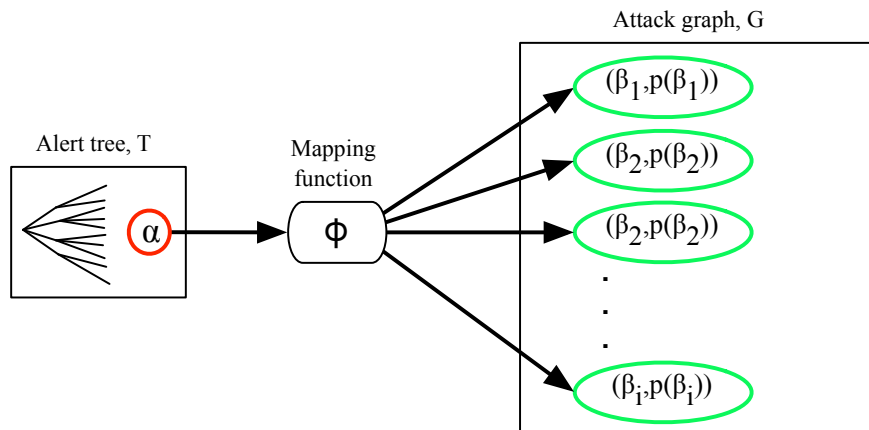


Figure 3.3: Anticipatory IDS architecture

In conventional approaches, the ordering of signature processing is often unrelated to previous events and truncating the pattern matching decision procedure can lead to loss of important events (false negatives which matter). Static prioritization of signatures can only partially resolve this problem and presumes that a given fixed signature prioritization is appropriate in all instances. In the new approach we can more safely truncate the decision procedure and potentially eliminate a significant portion of the processing. The amount of effort saved in such a case is proportional to the difference in the total number of signatures to the number of signatures in the equivalence class.

3.1.1 Wasted Information

Potential performance sinks in IDS systems can be understood in terms of wasted work. For example, work is wasted when an IDS engine duplicates signature processing, when it ignores correlations between outputs and future inputs, or when partial matches are discarded instead of being used to bootstrap future detections. Wasted work occurs in modern IDS systems. Some of these issues are solvable via simple caching mechanisms. Others require solutions with more sophistication.

Chapter 3. Theory

If there is no prior knowledge to guide the computing of signatures for a new network packet, then the entire decision procedure must be performed against the incoming data. This exhaustive matching cannot be considered as wasted work unless there were a direct way of improving its performance. However, if we have prior events then we can often modify the IDS detector to perform more efficiently, often with some loss in precision. We can measure the potential performance improvement by assessing how much effort is wasted by performing the exhaustive matching. Although not a conventional measure for work performed, we can think of work wasted in terms of bits of information. The work performed in most cases is related directly to the length of packets in bits and the complexity of signatures. If useful information is gained which is subsequently discarded, then this information has been wasted. In an anticipatory system the gains in IDS performance are directly proportional to the number of bits which no longer need to be processed by a particular decision procedure or IDS signature.

Ignoring Anticipation: In respect to our anticipatory model, work is wasted when we do not perform a prediction that we could have performed using the architecture. For decision procedure T we can define a function $\Omega_T : \{p \in P\} \rightarrow \mathbb{R}$, which calculates the information (in bits) used to process a particular packet given decision procedure T . When compared to a smaller decision procedure T' , the information which is wasted by the exhaustive approach for inputs p_0 to p_n is simply:

$$\Delta\Omega = \sum_{i=0}^n (\Omega_T(p_i) - \Omega_{T'}(p_i)) . \quad (3.1)$$

Note that we count information as wasted not when it is produced, but when an IDS fails to use it. If some particularly relevant information can be applied many times to decrease the size of a decision procedure then it we are wasting information for every instance where the information could have been used.

Duplicate Alerts: In an IDS with no knowledge of event and alert distributions, work is wasted whenever we produce duplicate alerts for the same connection. Computing a

Chapter 3. Theory

signature when we have already detected a particular event is costly and accounts for a small additional burden to the IDS. Duplicate alerts for the same IP connection means that the signature was computed multiple times after its initial detection. Every instance except the first can be considered wasted effort. If signature s is expected to fire alert α within the first k packets, then the wasted information in processing all packets in the set of packets, $p \in P$, comprising the entire sequence is:

$$\Delta\Omega = \sum_{i=k}^{|P|} \Omega_s(p_i) . \quad (3.2)$$

That is, the signature is processed $|P| - k$ times after it is initially detected, each instance being wasted information proportional to the size of the processed packets. The first k signature computations are necessary unless good statistics are known over where in the packet sequence the event is most likely to occur. If this knowledge is available for a particular alert, then additional optimizations would be possible and not using the optimization represents wasted work.

The Earliest Alert: Another form of wasted information which is related to duplicate alerts occurs due to our desire to produce an alert as early as possible within a sequence of packets. Some types of alerts are likely to occur many times if they occur at all. Nonetheless, an IDS generally tests every “qualified” signature against every packet within a packet sequence even when relatively infrequent signature tests would still discover the event. If our goal is to see the event at least once per packet sequence then we would test only 1 packet per n_α , where n_α is the number of times per sequence that alert α will occur if it occurs at all. Assuming we are randomly activating the signature with probability $\frac{1}{n_\alpha}$, the amount of wasted information is:

$$\Delta\Omega = \left(\sum_{i=0}^{|P|} \Omega_s(p_i) \right) - \Omega_s(p_{avg}) \frac{|P|}{n_\alpha} , \quad (3.3)$$

where p_{avg} is a packet with “average” characteristics for those within sequences which result in the alert. We only needed to test $\frac{|P|}{n_\alpha}$ packets but we tested all packets in the

Chapter 3. Theory

sequence $p \in P$. The optimization would allow many signatures to be tested with relative infrequency, but with an added expected latency of n_α packets. Further, if we have good distributional statistics over when events are likely to occur within a sequence then we may also not be giving up much in terms of latency. We can match the signature-activation probability for each packet with the likelihood of seeing the event at that time (or packet), optimally activating signatures to incur the smallest additional latency possible given the quality of our knowledge of the distributions. See Chapt. 6.3 for some experimental results relating to this class of wasted information).

Flow Tracking: One class of optimization which is almost always incorporated into modern IDS is flow tracking. This is a mechanism for tracking the state of individual connections such as the state of a TCP connection: whether it is established, its direction (client initiated or server initiated), and whether the connection has been closed and by whom. A system which did not track session state would likely require more complex signatures, be incapable of detecting some types of events without additional false-positives, and spend significantly more time processing signatures that would not be processed had the session state been known.

Alert and attacker action granularity mismatch: In an anticipatory model work can also be wasted when the decision tree and attack graphs are mismatched. If the tree and graph are perfectly matched then there is no aliasing of alerts to the same equivalence classes. However, there may exist similar alerts, $\alpha_1, \alpha_2, \alpha_3$ which are more precise alerts than α , i.e. $\{\alpha_i \Rightarrow \alpha\}$, the set of alerts which each independently imply alert α . If each α_i is capable of being mapped to a more precise set of graph vertices, then improving the decision tree may have been worthwhile for the purposes of improved predictions. However, if they all map to the same vertex, then we wasted effort in discriminating between the different alerts, increasing the total work performed.

With these more precise alerts, the function Φ maps alerts into sets of attack graph vertices B_i . If the vertices B_i are the same set as B_α then the improved granularity of the

Chapter 3. Theory

decision tree was information gained, but work wasted. The cost difference between the two represents the wasted effort which can be estimated by comparing the signature cost for α and α_i . When Φ aliases any of the alerts, then each aliased alert was wasted work performed by the decision procedure. Using again our information cost function Ω_s defined to assess the cost of matching against a single signature s , for k alerts which alias to the same prediction, for a single packet the information wasted is:

$$\Delta\Omega \leq \left(\sum_{i=0}^k \Omega_{s_i}(p) \right) - \Omega_s(p), \quad (3.4)$$

where s and s_i result in alerts α and α_i respectively, $\{\alpha_i \Rightarrow \alpha\}$, and $\Phi(\alpha_i) = \Phi(\alpha)$. The number of signatures which are aliased and their respective costs could be substantial. This is an overestimation as it is the degree to which $\Phi(\alpha_i)$ and $\Phi(\alpha)$ are the same that is important. Since these are sets, a more precise expression might be formulated.

It may not be clear, but this analysis makes a fairly bold claim concerning the purpose of an IDS. Currently an IDS does not generally concern itself with attacker intent. If determining current and future attacker intent were the purpose of an IDS, then any alert which maps to the same set of attacker actions as another alert would be duplicating effort.

Prediction distribution and variance: Wasted work can also occur in the mapping of the set of predicted actions into equivalence classes. Clearly, the fewer equivalence classes predicted, the better the resulting performance. The optimal performance occurs when we predict only a single equivalence class with absolute certainty. If we predict a large set of equally likely equivalence classes then we will not gain much improvement. In fact we may have wasted work in performing predictions which were not sufficiently helpful. The worst case is the uniform distribution in which every equivalence class is included in the set of predictions, each with equal likelihood. Work is wasted due to too many equivalence classes being predicted with a more or less uniform distribution. It is the ability of the system to differentiate between equivalence classes of signatures and alerts that enables an improvement in performance.

3.1.2 Threat Coverage

Many past research papers have loosely referred to the *coverage* of an IDS to encompass several different concepts which are often overlapping, namely: the portion of the entire attack space which is detectable by the IDS (attack-space coverage)[1, 25, 27, 34, 45]; the portion of network, hosts, or other objects which are processed by a detection system (sensor coverage)[6, 40, 44, 48]; and the portion of vulnerable operations or program paths which a given signature or other detection technique can discover (vulnerability coverage)[24, 23]. We can define a variant of attack-space coverage, *threat coverage*, to mean the portion of existing threats which can be discovered by an IDS. We might also choose to consider a threat both in terms of attacker/vulnerability and in exposure of information by either a victim or adversary. Exposure information is particularly relevant for anticipatory techniques (dealing with relevant characteristics of a potential victim such as Operating System, running services, software versions, etc.). However, host exposure information is not part and parcel to conventional IDS techniques (even though such use was clearly articulated by Denning in her general IDS model[31]).

As in Sect. 2.1.4 we define the *threat coverage*, A , of an IDS as the union of the sets of vulnerabilities A_{vuln} , exploits A_{exploit} , victim characteristics A_{victim} , and attacker characteristics A_{attacker} which are accurately identified (i.e. $A = A_{\text{vuln}} \cup A_{\text{exploit}} \cup A_{\text{victim}} \cup A_{\text{attacker}}$). Threat coverage (TC) can be defined in either relative or absolute terms. An absolute measure of coverage (Eq. 3.5) simply enumerates the expected number of properly identified events. We use expected value as not all threats in A can be detected with 100% confidence.

$$\text{TC}_{\text{absolute}} = |E(A)| . \quad (3.5)$$

An IDS or IDS configuration for which $|E(A)|$ is larger indicates better threat coverage (and/or better precision over the same threats). This is only a sample set of all possible identifications. Without knowing the true population of possible identifiable threats (A_{true}) the total number of known vulnerabilities, k_v , might be estimated from existing vulnerability

Chapter 3. Theory

databases. The expected number of exploits per vulnerability, k_x , within our sample set can be estimated from existing exploit databases or by examining available attack tools.

$$TC_{\text{true}} = |A_{\text{true}}| \simeq k_v * k_x . \quad (3.6)$$

The relative threat coverage is simply the ratio of relative coverage to the total number of estimated threats.

$$TC_{\text{relative}} = \frac{TC_{\text{absolute}}}{TC_{\text{true}}} = \frac{|A|}{|A_{\text{true}}|} . \quad (3.7)$$

Such a measure provides a rudimentary estimate of the extent to which the signature-set has absolute coverage of exploits. Threat coverage is independent of the dataset being input into a detector, and provides a rough estimate of the quality of the detection system.

3.1.3 Packet Coverage

In common IDS parlance, communication events which produce an alert are called *qualifying* and events which do not result in alerts *non-qualifying*. The number of qualifying events is often used as a metric to determine whether an IDS configuration is producing acceptable number of output alerts. Using this metric, however, it is quite possible to have a larger number of qualifying events than actual packets. However, what we need to be concerned with is the number of packets which result in *any* information being gained which could be used for prediction. For this, a different definition of coverage is needed.

The Snort IDS, as many others, only produces a useful output (a true positive) when a packet matches one of a relatively small number of patterns. As a results the *packet coverage* of these systems is a small fraction of the total packet data being processed, limiting the effects that any bias could have to a small fraction of the total traffic.

The coverage ratio of the IDS is the ratio of packets which result in identification of an item in A to packets which produce no information. For simplicity, we can restrict our analysis to events derived from individual packets, though there are many ways in

Chapter 3. Theory

which IDS produce events based on packet sequences (such as Snort's *stream5* preprocessor which allows events to be derived from a reassembled TCP stream [70]). For packets, p in the set of all packets processed, P , and an abstract packet feature matching function $F : p \rightarrow (\alpha \in A \cup \emptyset)$ representing the operation of the IDS (mapping packets into alerts, α or into *null* events), the packet coverage ratio A_p can be estimated as:

$$A_p = \frac{\left| \bigcup_{p \in P} (F(p) \in A) \right|}{|P|}, \quad (3.8)$$

where the numerator is the total number of packets which produce qualified events, and denominator is the total number of packets.

For a given configuration and signature-set of the Snort IDS, A_p can be estimated by running the IDS against a set of packet data and determining the ratio of packets which produce IDS events. The packet data used should cover a period of high traffic volume to preclude overestimation due to diurnal traffic patterns (e.g. low occurrence rates of normal traffic and relatively high occurrence of qualifying events). What we want is an underestimate of coverage. If an abnormal amount of attack traffic is already present, then this also may produce an overestimate.

A related measure, the predictor coverage A_q estimates the ratio of qualified events which can produce predictions at or above a desired confidence threshold t . Given a predictor, the probability of alert α_j given alert α_i is $P(\alpha_j|\alpha_i)$. The predictor coverage is the ratio of events which provide sufficient predictors to total qualified events.

$$A_q = \frac{\left| \bigcup_{p \in P} (F(p) = \alpha_j \in A \text{ where } P(\alpha_j|\alpha_i \in A) > t) \right|}{\left| \bigcup_{p \in P} (F(p) \in A) \right|}. \quad (3.9)$$

$$A_r = A_p A_q. \quad (3.10)$$

The combined coverage ratio, A_r , represents the proportion of packets which result in events and for which predictions would be made within an anticipatory system (i.e. meet desired thresholds on predictor confidence measures).

3.1.4 Signature Equivalence Classes

We can define an “equivalence class” of signatures to be a set of signatures which is relevant for a particular subset of packet traffic. For example, a set of signatures which relate to Microsoft-specific software and operating system components are relevant for the portion of packet traffic which is sent and received by computers which are running the Microsoft Windows operating system. Equivalence classes of signatures need not be disjoint. There are many instances where multiple taxonomic categories might be useful for defining signature equivalence classes.

Specifically the features that might be used as a basis for equivalence classes belong to the same set of features which are identifiable by IDS systems: vulnerability characteristics A_{vuln} , exploit characteristics A_{exploit} , victim characteristics A_{victim} , and attacker characteristics A_{attacker} . Without describing such features exhaustively, a broad interpretation of this set of characteristics can be considered to be exhaustive, covering all aspects of computer processing, storage, and computer-to-computer interactions.

Current threat taxonomies such as those by Herzog et al.[47] describe only a very high-level view of threats (representing only about 50 distinct classes in some cases). Although remarkably deficient in detail, these taxonomies can still be drawn upon for the definition of equivalence classes. Without defining new taxonomic relationships, these existing taxonomies can also be expanded by broadening the definition of equivalence class to be a tuple consisting of multiple characteristics, such as attack category and attacker intent: $\xi = \{(\alpha, \alpha_{\text{intent}}) \in A | (\alpha, \alpha_{\text{intent}}) \sim \xi\}$. Additional features can be added in this way to multiplicatively increase the size of the set of equivalence classes.

This potentially expands the number of available equivalence classes to several thousand. The difficulty with the inclusion of intent (or other characteristics) is knowing their association. Further, not all characteristics are reasonably associated with every type of attack, making the threat/intent associations sparse in practice. If there are no alerts asso-

Chapter 3. Theory

ciated with a particular tuple, then we gain no benefits for including the additional class. Expanding the number of equivalence classes in this way can be performed indefinitely as new characteristics are included (such as the host's exposure). The added burden is learning the associations between the expanded set of features in the tuple. Though many possible equivalence classes can be constructed, the limit in the application will be the availability of data and time required for learning.

For some set of equivalence classes, E and a set of signatures S , $k_\xi = |E|$ is the total number of equivalence classes. If there is one decision tree per equivalence class and only the maximum likelihood tree is processed, this directly determines the number of smaller decision trees which must be considered. If our initial signature-set size consists of all signatures, $n = |S|$, and all subtrees are approximately equal size, k_ξ equivalence classes would mean that on average n/k_ξ signatures must be considered for each predicted equivalence class (see table 3.1).

Not considering the performance of the predictor (or variations in the individual computing costs of signatures), the decrease in problem size, Δn , from the original, is simply:

$$\Delta n = n' - n , \quad (3.11)$$

where, the new problem size is $n' = \frac{n}{k_\xi}$ and $1 \leq k_\xi < |S|$.

The problem size (average size of resulting trees) and the number of equivalence classes are equal when $k_\xi = \sqrt{n}$. It is highly desirable to have the number equivalence classes grow slowly with the number of signatures. We see from Table 3.1 that having anything but a constant number equivalence classes results in average problem sizes which are small. This is not surprising, but generating large numbers of useful equivalence classes is not trivial.

An unrealistic upper bound on our improvement would take the extreme case when the number of equivalence classes is a constant fraction of the total number of signatures, $k_\xi = n/m$, where m is a constant. Maintaining a constant ratio of signatures to equivalence

$k_\xi = E $	n'	scaling (n')	scaling (k_ξ)
1	n	$O(n)$	$O(1)$
2	$n/2$	$O(n)$	$O(1)$
k_ξ	n/k_ξ	$O(n)$	$O(1)$
$\log n$	$\frac{n}{\log n}$	$O(\frac{n}{\log n})$	$O(\log n)$
\sqrt{n}	\sqrt{n}	$O(\sqrt{n})$	$O(\sqrt{n})$
n/m	m	$O(1)$	$O(n)$
$n/2$	2	$O(1)$	$O(n)$
n	1	$O(1)$	$O(n)$

Table 3.1: Equivalence class size and problem size

classes would result in $O(1)$ scaling performance for individual tree traversals. However, achieving constant-time performance would currently be challenging due to the desire to process very large signatures sets (thus requiring a proportionally large number of equivalence classes). More realistically, the total number of equivalence classes will be limited and an anticipatory approach will rely on the packet coverage of a set of signatures. In reality, neither the sizes of equivalence classes, the costs of individual signature computations, nor the quantity of packet traffic which is relevant to the class are uniformly distributed. So long as the distributions can be learned, this is generally desirable. If a disproportionate portion of packet traffic belongs to an equivalence class containing a smaller number of signatures, then better gains can be achieved than in cases where equivalence class characteristics are uniformly distributed. Indeed, one straightforward mechanism for decreasing IDS costs is to make use of signature cost and alert occurrence distributions directly (see Chapt. 3.2).

3.1.5 An IDS Cost Function

Although the true cost functions for performing detection using a given IDS configuration and computing system is unlikely to be known precisely, they can be estimated using experimental measurements. We require this cost function in order to determine the conditions necessary for anticipatory bias to achieve an improvement to performance.

Chapter 3. Theory

In particular, we can represent this problem as a function representing performance gain, C_g , incorporating cost functions C_{time} (CPU time) and C_{loss} (packet loss) and parameterized by: coverage ratios A_p and A_q ; signature-set size, $n = |S|$; and average secondary signature-set size $k = |E|$. The cost functions can be multiplicatively combined to mean CPU-time per packet, appropriately penalizing high CPU-time or high packet loss. Note that both C_{loss} and C_{time} are ratios. If C_{loss} is the fraction of packets which are dropped, then $1 - C_{\text{loss}}$ is the fraction which is processed. We can use CPU-time and packet loss costs as a single cost function:

$$C(n) = C_{\text{time}}(n)C_{\text{loss}}(n) = \frac{C_{\text{time}}(n)}{1 - C_{\text{loss}}(n)}. \quad (3.12)$$

We define gain, C_g , as the ratio of the costs between a system running without a predicted equivalence class of events and one which utilizes an anticipatory approach. The cost of the IDS running in a non-anticipatory mode is simply $C_{\text{primary}} = C_n$ for a primary signature-set of size n . The cost of the secondary IDS instance for an anticipatory system is: $C_{\text{secondary}} = A_r \cdot C_k$, where A_r is the proportion of events which are expected to be solely processed by the secondary IDS instance. In a complimentary fashion, after initial bootstrapping, the cost of the primary IDS instance would be $C_{\text{bootstrap}} = C_n(1 - A_r)$ as an (A_r) fraction of the traffic is shunted to the secondary instance. The performance gain in terms of packets per unit CPU-time is then:

$$C_g(A_r, n, k) = \frac{C_{\text{primary}}}{C_{\text{bootstrap}} + C_{\text{secondary}}} - 1 = \frac{C_n}{C_n(1 - A_r) + C_k \cdot A_r} - 1. \quad (3.13)$$

Any gain larger than 0 is a performance improvement. A performance difference, ΔC , can also be defined which gives an absolute measure of the improvement or degradation in terms of packets processed per unit of CPU-time:

$$\Delta C(A_r, n, k) = C_{\text{primary}} - (C_{\text{bootstrap}} + C_{\text{secondary}}). \quad (3.14)$$

Which can be written using the cost function C as:

$$\Delta C(A_r, n, k) = C_n - (C_n(1 - A_r) + C_k \cdot A_r)$$

$$\Delta C(A_r, n, k) = A_r(C_n - C_k). \quad (3.15)$$

For simplicity and without loss of generality, the cost function of a system with only a single secondary IDS instance is considered. Extending the cost function definition to describe multiple predicted equivalence classes is straightforward.

3.2 Probabilistic Signature Activation

For high-coverage signature-sets it is necessary to minimize the number of times that any particular signature is activated (while simultaneously maintaining coverage). This is particularly important for non-alert signatures used for gathering attacker and victim characteristics (such as host exposures). By definition, high-coverage signatures will occur within a high percentage of the input packets. If a signature is being computed and fires an alert many times per stream, it is likely to be wasting information in terms of duplicate alerts (see Sect. 3.1.1 on page 58). Indeed, results using the Packet Wrangler prototype with high-coverage signatures show that anticipatory gains are outweighed by the additional cost of processing alerts which fire for a large portion of the input traffic (see Chapt. 6.2 on page 119).

If the event associated with a signature occurs frequently, then we should check for it infrequently. Indeed, the more frequently an event is likely to occur, the fewer computing resources we need to assign.

The goal is to limit wasted resources in order to improve detector performance. Our current approach is to trade some number of carefully chosen false negatives for improved performance overall. As such we only care about detecting at least one instance of every unique event for every tracked packet stream which actually contains the event. It is intuitive that if such an event occurs frequently, and we only care about seeing it once for a given time-span, then we should check for it infrequently. Detecting a larger number than this is

wasting computing resources. In this context, the more frequently an event is likely to occur, the fewer computing resources we need to assign. The initial intuition for this approach is based on rudimentary probability and statistics. For a given network or dataset, we can collect statistics for each event α to learn the number of times we expect to see the event for each stream. If we expect to see α , n_α times out of the n_p packets in the stream, we will still have a *good* chance of detecting the event while minimizing signature computing costs by activating the associated signature, s , with probability:

$$P(\text{activate}(s)) \geq \frac{n_\alpha}{n_p} . \quad (3.16)$$

If the signature activation probability is too large, then we waste effort in making multiple detections of the same activity. If the signature activation probability is too small, then we will miss events. The trade-off being made is two-fold.

Firstly, we are choosing to define a false negative as having missed *all* of the events α in the stream. This is consistent with the suppression mechanisms of existing IDS, often suppressing all by a single event within the stream for a given time window.

Secondly, we will incur some additional latency between when the event actually occurs and when it is detected. If we were suppressing an event at output we would still guarantee that we see the first of such events. Although our approach does not use detailed distributional statistics of events over streams, a more sophisticated approach could vary signature activation probabilities based on where in the stream an event was likely to occur, limiting (and in some cases eliminating) the effects of increased detection latency. The goal of controlling for and limiting detection latency is left to future studies.

3.2.1 Signature Activation Policies

It is fairly straightforward to define a set of signature-activation policies to decrease the number of times that a signature is used. Each policy results in varying performance characteristics. The policies that have been studied combine measurements of: signature cost,

Chapter 3. Theory

alert relevance (i.e. frequency), and system load. Each policy is able to decrease amortized per-signature processing costs by independently activating each signature with a probability inversely proportional to the policy parameter for the signature (i.e. cost, frequency, load, or combinations thereof). Interestingly, these policies can achieve performance gains without incurring additional errors when the system is overloaded and dropping packet data (often due to CPU-contention between I/O operations and detection engine processing). The following describe possible signature activation policies, each of which can be used in a deterministic or non-deterministic fashion (i.e. having either a discrete value threshold or a probability as an activation criteria respectively). Deterministic rule-chaining is the only policy currently implemented within the standard Snort distribution.

Signature Chaining Policy

A *signature chaining* policy guarantees signature activation according to non-probabilistic semantics with prior signature alerts as preconditions.

$$\text{fire}(s_a) \rightarrow N_p(\text{activate}(s_b)) , \quad (3.17)$$

where s_a is an active signature, s_b is an inactive signature with the precondition $\text{fire}(s_a)$, and N_p is the *Next* temporal semantics operator defined over packets (see Table 2.1 on page 35). Whether or not a signature is activated immediately for all signatures which have yet to be tested for the current packet depends on the design of the IDS and activation mechanism. At least within the Snort IDS, there is no guarantee that signature activation will occur within the processing of a single packet. Signature activation is only guaranteed for all future packets within the same stream[79].

Cost Policy

A *cost* policy chooses a signature activation probabilities which are inversely proportional to expected signature computing costs.

$$P(N(\text{activate}(s_a))) = \max \left(1 - \frac{\text{cost}(s_a)}{\underset{s \in S}{\text{argmax}}(\text{cost}(s))}, P_{\min} \right), \quad (3.18)$$

where $\text{cost}(s) = \Omega_s(p_{avg})$ is the average cost for computing signature s_a and a minimum activation probability, P_{\min} , is chosen.

Relevance Policy

A *relevance* policy chooses signature activation probabilities which are inversely proportional to expected signature firing frequencies.

$$P(N(\text{activate}(s_a))) = \max \left(1 - \frac{\text{count}(s_a)}{\underset{s \in S}{\text{argmax}}(\text{count}(s))}, P_{\min} \right), \quad (3.19)$$

where $\text{count}(s)$ is the average per-session alert counts for signature s .

Load Policy

A *load* policy chooses signature activation probabilities which are inversely proportional to system load.

$$P(N(\text{activate}(s_a))) = 1 - \frac{\text{load}}{1 + P_{\min}}, \quad (3.20)$$

where *load* is a proportion of total system overhead, $0 \leq \text{load} \leq 1$. A non-linear activation policy might use any appropriately decreasing function defined over $f : [0, 1] \rightarrow [0, 1]$ designed to map system loads to probabilities, such as the function shown in Fig. 3.4.

$$P(N(\text{activate}(s_a))) = 1 - P_{\min} - (1 + P_{\min}) \left(\frac{P_{\min}}{1 + P_{\min} - \text{load}} \right). \quad (3.21)$$

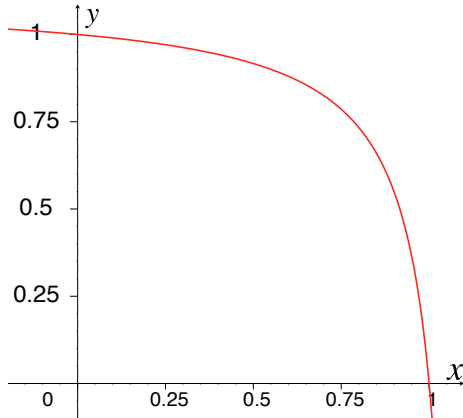


Figure 3.4: Probabilistic signature activation function
Function maps system load to activation probability.

Cost-Relevance Policy

A *relevance* policy chooses signature activation probabilities which are inversely proportional to expected signature cost divided by expected signature firing frequency.

$$P(\text{activate}(s_a)) = \max \left(1 - \frac{\text{costFun}(s_a)}{\underset{s \in S}{\text{argmax}}(\text{costFun}(s_a))}, P_{\min} \right), \quad (3.22)$$

where $\text{costFun}(s_a) = \text{cost}(s_a)/\text{count}(s_a)$. This policy is the one initially considered within experimental configurations as we felt it would provide the most generalization between different time windows. It also maps directly to the output of Snort's detection engine profiler discussed in Sect. B.2.

Combined and Multiple Measure Policy

It should be obvious that these signature activation policies can be easily combined to make a signature's activation dependent upon combined measures or multiple independent measures.

3.2.2 Probabilistic Flowbits and False Negatives

Using a probabilistic signature-activation policy when a system is easily keeping up with packet traffic will decrease system overhead, but results in increases in error rates (see Chapt. 6.3 on page 133). However, when even a moderate loss in packets is present, each of the studied policies performs better than a system with the default policy, decreasing both system overhead and error rates. The reasons for performance gains is interesting and an analysis of signature activation policies and expected error rates provides useful insights.

In the Snort IDS the default Flowbit implementation is used for two principal purposes: a) to track state over the extent of a TCP session; and b) to prevent high-cost signatures from being activated needlessly. The end result both expands the capabilities of the IDS and decreases the average cost of signatures which are “hidden” behind flowbits (assuming an alternative “non-Flowbit” implementation was possible).

Consider two signatures, s_a and s_b , and their respective alerts, α_a and α_b (as in Fig. 3.5). If s_b checks a Flowbit set by s_a when an event, α_a is emitted, then the expected cost of

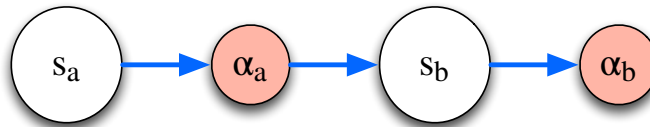


Figure 3.5: Simple state transitions for Flowbit-based signature activation

signature s_b will be lower than a hypothetical setup which did not rely on a Flowbit. For the default Flowbit setup, the expected cost difference between hiding the signature versus allowing it to always be active is proportional to the simple probability of event α_a :

$$\Delta C = P(\alpha_a)\Omega_{s_b} - \Omega_{s_b} = (P(\alpha_a) - 1)\Omega_{s_b} , \quad (3.23)$$

where Ω_{s_b} is the expected per-packet cost of computing the signature.

Chapter 3. Theory

If event α_a is rare, then very little cost will be incurred in processing signature s_b . However, any session which activates the Flowbit will do so for the duration of the session. As a result we may end up with many activations of s_b for each session and produce many alerts when a single activation would have been sufficient. The new signature-activation policies allow us to optimize signature processing costs based on these new considerations.

With the proposed signature activation policies we are choosing to decrease the burden imposed by some subset of signatures which we believe are either too costly or too frequent. If the chosen probability for signature activation of a signature s_b is $P(s_b)$ then the expected cost difference is:

$$\Delta C = (P(s_b) - 1)\Omega_{s_b} \quad (3.24)$$

Of course, the only reason for using flowbits in this manner is to hide high-cost signatures when system loads are causing packet loss. Otherwise, while we will surely decrease system overhead we will also incur additional errors.

For the default policy, the expected number of false-negatives in respect to a single signature when there are n packets and packet loss rate of C_{loss} can be stated simply as:

$$FN(\alpha_i) = P(\alpha_i)nC_{loss}(S) , \quad (3.25)$$

where $C_{loss}(S)$ is the cost function from Sect. 3.1.5 defined to return the ratio of packets which are dropped by the IDS when using a specific signature-set S for a given system and configuration. As such, nC_{loss} is the total number of packets which are dropped when n packets are processed by signature-set S . If event α_i occurs with probability $P(\alpha_i)$ then Eq. 3.25 is the expected number of dropped packets which would have resulted in event α_i .

These expressions make a simplifying assumption that each packet is equally likely to be dropped. There are two potential issues with this.

Firstly, if packets were being dropped from a TCP handshake, then a larger number of false negatives would occur for the connection in question. The IDS would not be able

Chapter 3. Theory

to track the TCP stream and would therefore be unable to process a significant number of potentially applicable signatures for the duration of the TCP stream. This issue is described in Papadogiannakis et al.[71] where they present a possible solution which selectively drops packets which are not crucial for session tracking. In our case, if a signature requires session tracking the signature will be inactive for the affected sessions. When such crucial packets are dropped, we may as well have dropped a series of packets from the same session. Fortunately, we can argue that the average number of packets per session is much larger than the number of session initiation packets. For instances where this is not true (as is the case for some types of connections), session tracking is not as important for detection. So, while the impact of losing crucial packets is larger, the likelihood of losing these packets is generally quite small, particularly when datasets are large.

Secondly, datasets which represent only a small number of connections may result in biases in which packets are dropped. In such cases, packets for the same connection may occur bundled together in the incoming queue of the IDS. If the IDS process is busy and its incoming queue blocks, packets will be lost. This could result in drops of substantial portions of single connections. For larger datasets (representing many thousands of connections per minute), packets which are currently queued are unlikely to contain such bundles. The packets dropped, while representing a set of packets which is correlated in time, would be unlikely to be correlated *spatially* across the network. Just as in the case of losing session initiation packets, the larger the dataset (meaning the larger the number of simultaneously active connections), the more mixed the incoming queue will be.

It is important to reiterate that we are discussing IDS which capture streams of traffic from network gateways, routers, or hubs and as such are processing data from as many simultaneous connections as are currently being routed across the network. For an IDS which resides at a single (or limited number of hosts) these arguments are not valid.

Chapter 3. Theory

Returning to our discussion of false negatives, the expected number of false negatives for all signatures is simply the sum over all signatures:

$$\text{FN}_{\text{default}} = \sum_{\alpha \in A} P(\alpha) n C_{\text{loss}}(S) . \quad (3.26)$$

Alternatively, the number of false negatives for either of the *relevance* or *cost* policies differs due to a subset of signatures only being active with a chosen probability of $P(s_i)$ and thus inactive with probability $1 - P(s_i)$.

The new false negative rate for the new activation policies for event α_i is summarized by Eq. 3.27.

$$\text{FN}(\alpha_i) = nP(\alpha_i)C'_{\text{loss}} + nP_{\alpha_i}(1 - P(s_i))(1 - C'_{\text{loss}}) , \quad (3.27)$$

where C'_{loss} is the new packet loss rate after signature s_i is removed; “ $nP(\alpha_i)C'_{\text{loss}}$ ” describes the number of packets dropped that would have resulted in event α_i (as in Eq. 3.25); and the term “ $nP(\alpha_i)(1 - P(s_i))(1 - C'_{\text{loss}})$ ” describes the number of packets processed which contain event α_i but for which signature s_i is inactive. The new packet loss rate, C'_{loss} , estimates the packet loss when a smaller signature-set is used with probability $(1 - P(s_i))$ and can be estimated as:

$$C'_{\text{loss}} = C_{\text{loss}}(S)P(s_i) + C_{\text{loss}}(S')(1 - P(s_i)) , \quad (3.28)$$

where S' is the set of signatures excepting the signature which was removed, $S' = \{s \in S | s \neq s_i\}$. We will not expand C'_{loss} in our current analysis but define it here for clarity. While this term is somewhat impractical to determine analytically due to myriad factors related to signature complexity and muddy IDS implementation specifics, it is relatively straightforward to determine experimentally for a given signature-set and dataset (see Sect. 4.2.4 on page 97).

We can use Eq. 3.27 to show that under certain conditions we can decrease false negatives for the signature being held inactive. We can calculate a differences in false negatives as $\Delta\text{FN} = \text{FN}_{\text{new}} - \text{FN}_{\text{default}}$. If $\Delta\text{FN} < 0$ then we will have decreased false

Chapter 3. Theory

negatives for signature s_i . It is important to note that Eq. 3.27 accounts for all false negatives, even those that are duplicate events. Since we only care about whether the false negative rate differs between the default policy and the new policies, this is not an issue. Rewriting Eq. 3.27 and re-using Eq. 3.25:

$$\text{FN}_{\text{new}}(\alpha_i) = nP(\alpha_i)[(C'_{\text{loss}} + (1 - P(s_i))(1 - C'_{\text{loss}}))] .$$

Combining terms and simplifying, we get:

$$\text{FN}_{\text{new}}(\alpha_i) = nP(\alpha_i)[1 - P(s_i) + P(s_i)C'_{\text{loss}}] . \quad (3.29)$$

We can now write a simple expression for $\Delta\text{FN}(\alpha_i)$:

$$\Delta\text{FN}(\alpha_i) = nP(\alpha_i)[1 - P(s_i) + P(s_i)C'_{\text{loss}}] - nP(\alpha_i)C_{\text{loss}} ,$$

which simplifies to:

$$\Delta\text{FN}(\alpha_i) = nP(\alpha_i)[1 - P(s_i) + P(s_i)C'_{\text{loss}} - C_{\text{loss}}] . \quad (3.30)$$

If $[1 - P(s_i) + P(s_i)C'_{\text{loss}} - C_{\text{loss}}] < 0$ then we would have decreased false negatives for signature s_i .

$$1 - P(s_i) + P(s_i)C'_{\text{loss}} - C_{\text{loss}} < 0 .$$

$$1 - C_{\text{loss}} < P(s_i)(1 - C'_{\text{loss}}) .$$

$$P(s_i) > \frac{1 - C_{\text{loss}}}{1 - C'_{\text{loss}}} . \quad (3.31)$$

The terms $1 - C_{\text{loss}}$ and $1 - C'_{\text{loss}}$ are the portion of packets processed when using signature-sets S and our probabilistic choice between S and S' respectively. In other words, if the probability of signature s_i being activated is greater than the ratio of packets processed in either configuration, then we will have decreased false negatives. What this means is that as long as we have a significant decrease in packet loss there is some signature activation probability that will decrease the false negatives for the signature in question.

Chapter 3. Theory

However, if the decrease in packet loss is small, then $P(s_i)$ will be close to 1 and unknown factors may result in increased false negatives. And if we choose a signature activation probability lower than the ratio of packet losses we will incur additional false negatives for event α_i . While this seems undesirable, it is likely that for some signatures we would gladly accept additional errors for a single signature as long as we can decrease error rates for the system as a whole.

So, for the signature which is being probabilistically activated, we can affect the number of false negatives depending on $P(s_i)$ and we can decrease total packet loss as a result of activating the signature less frequently. It is not obvious at first glance, but this represents only a small fraction of the potential improvement in false negatives. Because we are decreasing the number of lost packets, every signature in the entire signature-set will be compared against a larger portion of the incoming packets and may result in substantial improvements in false negatives as a result. Excluding α_i , the expected number of false negatives for the signature-set can be stated as:

$$\text{FN}_{\text{new}} = \sum_{\alpha \in A | \alpha \neq \alpha_i} P(\alpha) n C'_{\text{loss}} . \quad (3.32)$$

The difference in false negatives for the entire signature-set excluding s_i and its associated alert α_i becomes:

$$\begin{aligned} \Delta \text{FN} &= \sum_{\alpha \in A | \alpha \neq \alpha_i} P(\alpha) n C'_{\text{loss}} - \sum_{\alpha \in A | \alpha \neq \alpha_i} P(\alpha) n C_{\text{loss}} \\ \Delta \text{FN} &= (C'_{\text{loss}} - C_{\text{loss}}) \sum_{\alpha \in A | \alpha \neq \alpha_i} P(\alpha) n . \end{aligned} \quad (3.33)$$

The summation in Eq. 3.33 is total expected number of alerts when n packets are processed (excluding α_i). Clearly, the absolute decrease in FN can be substantial since we have an effect for every potential alert that will be produced by the IDS. Essentially we are trading possible increases in false negatives of α_i for *guaranteed* decreases in false negatives for all other events. Since it is unlikely that we care so much for α_i in particular, this is usually a very good trade-off. If we wanted to guarantee decreases in false negatives for all alerts,

Chapter 3. Theory

including α_i , we would simply limit $P(s_i)$ according to 3.31. Of course, we probably do not mind losing some of the α_i alerts as long as we have acceptable gains for other alerts.

An Unexpected Trade-Off

Experimentally, probabilistic signature activation also has a very non-intuitive side effect. For a system which is still dropping packets, when there are fewer packets which actually match signature s_i than expected, the approach will achieve better performance gains and fewer false negatives. We expend fewer resources by not looking for what is not there.

We have not yet discussed signature preconditions beyond their introduction in Chapt. 2.3.3 as they have not been particularly germane. In brief, many implementations bundled signatures into groups based on shared high-level attributes related to IP and TCP header fields such as IP address, port numbers, TCP flags, etc. These preconditions are very inexpensive to compute and provide a top-level decision of whether a bundle of signature-to-packet comparisons will be computed at all. If packets do not meet the preconditions for a bundle of signatures, then none of the signatures in the particular bundle will be computed.

When using the *relevance* policy, we base our decision to remove signature s_i based on the expected number of occurrences of the associated alert α_i within the current time window. In other words, we will not look for matches for s_i as diligently if we expect to see it often. Future time windows may contain a larger or smaller number of packets which would match s_i . If future time window has a larger portion of packets matching signature s_i , this implies that a larger portion of packets match all of the preconditions for activating the signature.

For the new signature activation policies, each time all of the preconditions are matched, s_i will be probabilistically activated. So a larger portion of matching packets will result in a larger number of signature-to-packet comparisons. As a result, any gains we have achieved

Chapter 3. Theory

in prior time windows will be diminished. The IDS is performing more work because a larger number of packets match the preconditions of signature s_i .

However, since we are controlling activation of a signature that is abnormally active and as many attacks are often short-lived, there will often be fewer matching packets in future time windows. This will result in fewer packets matching the preconditions for the signature and will subsequently improve any gains which were achieved. In other words, if the attacker is not attacking *and* we are not expending resources attempting to detect the attack, we will improve performance (including decreases in false negatives).

This explanation can be summarized:

1. We no longer incur as many false negatives for the signature in question as the event occurs less frequently.
2. Fewer packets match the preconditions required for the signature to be computed, so we do not attempt to activate the signature as often and this results in proportional decreases in computing costs.
3. The decrease in system overheads due to 2) results in more time available for processing additional packets and the false negatives decrease for the entire signature-set.

The reason that this is missing from our expression for ΔFN is that it is hidden within the loss function C_{loss} , which, as written is not dependent on the number of occurrences of α_i . The term would be more correctly defined to include the number of events, i.e. $C_{\text{loss}}(S, \text{count}(\alpha_i))$. Further, for the same number of packets we can write:

$$C_{\text{loss}}(S', \text{count}(\alpha_i)) < C_{\text{loss}}(S, \text{count}(\alpha_i)) < C_{\text{loss}}(S, \text{count}(\alpha_i) + 1) . \quad (3.34)$$

Fewer numbers of alert α_i will simply widen the gap between $C_{\text{loss}}(S')$ and $C_{\text{loss}}(S)$. It is possible that this effect can be leveraged for additional performance gains in some scenarios. However, it is likely that diminishing returns would result from happenstance decreases in

event frequency. It seems generally desired to minimize this effect by performing online measurements and dynamically updating signature activation probabilities to optimize system performance when under load. This effect and its implications should be more thoroughly explored, but this is left to future research.

3.3 Predictor Errors

The previous section described how probabilistic signature activation decreases false negative error rates under certain conditions. However, most uses of anticipatory approaches (such as when a system is not under heavy loads) will generally result in additional errors. These errors can be both false positives and false negatives. In general, anticipatory optimizations result in false negatives due to packets being processed against subsets of the original signature-sets. However, some types of signature chaining and anticipatory mechanisms can also result in false positives even though the signature-set is only ever being decreased in size.

3.3.1 Signature Chaining & Flow Tracking

Signature chaining is most often used to activate a signature which depends upon prior signatures, as in: *if fire(s_a) then activate(s_b)*.

Two signatures which have a negated relationship can result in a false positive if s_a is absent from the secondary signature or if packet forwarding to secondary IDS instances have disabled session tracking. For example, a false positive will occur within a secondary IDS instance if s_a is not part of the secondary IDS signature-set and there is a signature with negated chaining semantics: *if not fire(s_a) then activate(s_b)*.

Chapter 3. Theory

If flow-tracking is disabled entirely in a secondary signature-set can also result in error related to the direction of an established connection or the firing of alerts which relate to closing sessions which are not known to be established. To alleviate issues in signature-chaining caused by signature-set partitioning we must both a) keep chained signatures together when signature-sets are partitioned; and b) redirect packets to secondary sets which require as preconditions a particular flow state. For example, if the signatures used for prediction in the primary signature-set, $s_i \in S$, properly characterize flow, then we can infer that the flow is established and in which direction for any secondary signature-set ($s'_i \in S'$). It is relatively straightforward to partition secondary signature-sets into “to_server” and “to_client” sets and craft predictor rules to forward to the correct secondary instance. However, if the event that initiated packet forwarding did not track the session state for the connection then flow state of future packets will be unknown. Similarly, if any of the signatures in S' signature-sets has flow-tracking with negated preconditions, applying packets for connections with unknown flow state can result in false positives.

3.3.2 Systematic Errors

It is also possible to introduce systematic increases in false-negatives. This is particularly relevant when non-probabilistic or rule-based forwarding choices are made. If an attacker were to know the rule-based forwarding choices being made then it would be an easy target for circumventing detection. Rule-based predictors should be used only in instances where there is high confidence that the information being used to initiate packet forwarding is trustworthy. Alternatively, random testing of rule-based predictor assumptions (by disabling predictive forwarding) can help determine if an attacker is circumventing detection through misuse of predictor choices.

For the Bayesian predictor discussed in Sect. 5.1.3 a fixed confidence threshold can also incorrectly classify traffic due to short-term biases in traffic and sequence distributions.

Chapter 3. Theory

Increasing the window-size over which statistics are collected can help to minimize this issue, but would likely result in poorer performance. The problem with fixed thresholds and short time windows is that statistics are unlikely to be correct outside of the time window in which they were calculated. A possible solution is to perform online learning of attack and event distributions and of appropriate decision boundaries. Unfortunately, online learning can also lead to systematic errors, both unintentional and attacker-directed. Another approach may be characterizing various event distributions and using distribution change detection methods between sets of known distributions. This may result in less susceptibility to attacker-directed biases and represents a problem that has been studied heavily in the statistics literature[74, 50].

3.3.3 Error Detection

Error detection is a non-trivial task for detection systems. Several interesting approaches have been proposed and implemented by others[56], though many of these are only relevant when system loads are high enough for the IDS to drop incoming packets. Similar approaches can be taken to estimate errors for IDS event predictors. Lee et al. describe an approach which injects traffic into the system to determine the rate at which packets are being dropped[56]. This approach could be easily extended to inject packets which are intended to create an alert. Missed alerts can be used as a proxy to estimate IDS misclassification rates and to adjust predictor threshold accordingly.

If a fixed threshold is used then traffic which meets the criteria will always be forwarded and errors cannot be detected. If a probabilistic choice is made that is based on the current threshold, then packets which would have failed the criteria can be tagged in order to measure whether an error would have been made had they passed the criteria.

3.4 The Detection Game

Performance tuning of signature-sets has relied on signatures being always enabled or never enabled and removed from the detection engine entirely. High cost signatures are the first ones to be eliminated via this heavy-handed “tuning”. This approach will cause those attacks which were previously identified by the removed signature(s) to be undetectable. If an IDS administrator has a guarantee that a particular activity could never occur on their network (because of non-routable protocol, firewall configurations, removal of obsolete software or versions, etc), then removing signatures may be a reasonable approach. The *unexpected trade-off* discussed in the last section even suggests that when we know an event will not occur, then we should always remove the associated signature. That is, we get a better performance gain when we remove a signature for an event which is absent than when we remove a signature for an event which still occurs with some frequency.

In other cases, where there is some possibility of the event, removing signatures for purposes of performance tuning seems ill-advised. Further, it should be obvious that there are many possible trade-offs which can be leveraged to achieve better performance while maintaining better coverage of potential threats. In particular, as the previous analysis has shown, we can always do better than the default policy when there is any amount of packet loss.

Another valuable approach which is entirely missed by heavy-handed IDS tuning is one in which detection (and attack) are treated as a game in which a defender can choose whether or not to detect and an attacker can choose whether or not to launch an attack. The simplest type of game which we might analyze is a two player zero-sum game with perfect information. While all such games are well known in the sense of being exhaustively characterized, we can explore such a game to give ourselves a different intuition on how and why we would want to make choices between detecting and not detecting.

Chapter 3. Theory

		Defender		
		α	$\bar{\alpha}$	
Attacker	α	$-1 / 1$	$1 / -1$	q
	$\bar{\alpha}$	$-1 / 1$	$0 / 0$	$1-q$
		p	$1-p$	

Figure 3.6: The Detection Game payoff matrix for a zero-sum game

We define a two player game between an attacker and a defender in respect to a single type attack. The attacker’s purpose is to attack without being detected and to cause the detector to waste resources. The attacker desires a situation in which they can cause the largest amount of work for the defender for each attack performed (or not performed) so that unrelated attacks can go undetected due to the defender being overburdened. The defender’s purpose is to detect an active attack, but not to waste resources in attempting to detect activities which are not currently occurring. The defender has limited computing resources and desires to ensure that packets (and thus events) are not dropped due to being overburdened. Strategies and payoffs for the simplest of such games is shown in Fig. 3.6.

In this game the adversary is penalized for attacking while being detected and rewarded for attacking while not being detected. The defender is penalized for detecting while not being attacked and rewarded for detecting while being attacked. The logic here is that an adversary wants to either attack undetected or cause the defender to waste resources and the defender wants to detect attacks and not to use resources needlessly.

By inspection, this game has no pure Nash equilibrium. There is no state in which one of the players would not choose to change their decision and get a better payoff. Because this is a zero-sum game, there exists a mixed Nash equilibrium which is straightforward to compute using von Neumann’s minimax theorem. If the adversary will choose to attack with probability p , the expected utilities for the defender are $\alpha : (1)p + (-1)(1-p) = 2p - 1$

Chapter 3. Theory

and $\bar{\alpha} : (-1)p + (0)(1 - p) = -p$. The adversary can minimize the maximum payoff of the defender when $2p - 1 = -p$, so $p = \frac{1}{3}$. Similarly, if the defender chooses to detect with probability q , the expected utilities for the adversary are $\alpha : (-1)q + (1)(1 - q) = 1 - 2q$ and $\bar{\alpha} : (1)q + (0)(1 - q) = q$. The defender can minimize the maximum payoff of the defender when $1 - 2q = q$, so $q = \frac{1}{3}$.

To anyone familiar with basic game theory, this result is hardly surprising. However, it suggests that we should reflect on our current strategies for performing detection. The game described assumes very little about the differences between adversary and defender rewards and penalties. It suggests that if we know little about the penalty for not detecting, the cost of detection, or the rewards of the adversary (other than their sign) that we should only enable our detector $\frac{1}{3}$ of the time.

There are few obvious issues with this rudimentary analysis. Firstly, we have arbitrarily set payoffs. In a real-world scenario these are unlikely to be so simple or equitable. Such payoffs are also very difficult to know precisely, but there are some rough measures that could be used in regards to impact, processing costs, attack difficulty, etc.

Secondly, we assume that each strategy is zero-sum. It is unlikely that any “game” played between adversary and defenders is zero sum. Although for more complex games it might be necessary to make this assumption for tractable analysis. The actual rewards and penalties are clearly dependent on the type of attack, the type and purpose of the computer system being attacked, and myriad other factors. For some attacks, the adversary could care less whether they are detected (scans), and may not care about our processing costs because the attack is both inexpensive to detect and inexpensive to launch. Other situations would require high penalties both for an adversary being detected and for a defender detecting pointlessly (e.g. data exfiltration).

Finally, we have simplified things to deal with a single type of attack. In reality, the adversary has many possible attacks and the defender cannot detect all of them, making

Chapter 3. Theory

moot any analysis of whether or not to detect. Attacks are also often coordinated and have an affect that is mutually beneficial to the adversary.

It may be difficult to justify actual policy or IDS implementation changes based on an analysis which makes so many gross assumptions. Nonetheless, the beauty of using a game theoretic approach is that all of these situations can be modeled, along with situations with many attackers, many defenders, and many different attacks. While prior research has modeled aspects of intrusion detection in a game theoretic fashion[2, 7, 64], there is oddly very little research which assess detection choices in the context of a game. It seems crucial that we begin to seriously examine the potential trade-offs in detection systems. Failing to do so will result in environment which continues to lose the battle against smart and rational adversaries. The choice seems to be between playing the game or losing by forfeit.

Chapter 4

IDS Performance Characteristics

In order to determine whether the anticipatory model is applicable to improvements in the Snort IDS, the cost functions C_{time} and C_{loss} , needed to be experimentally determined. In general, the cost functions should represent conservative estimates of the actual performance costs, underestimating C_{primary} and overestimating $C_{\text{secondary}}$. For a given signature-set, the coverage ratio, A_p must also be learned.

It should be noted that the measurement approach obtains functional forms of the cost functions for a particular system and IDS configuration. Many of the performance characteristics of IDS systems are highly dependent on system and software configuration. Even small changes in configuration or run-time options can have significant effects on overall performance[81]. It is not our purpose to explore the entire parameter space of the Snort IDS, but to analyze the applicability of anticipation for improving performance and to compare performance with and without anticipatory bias. The hardware and software used for these and other experiments is described in Appendix A.1 on page 156. Hardware configuration, system settings, and software configurations were kept consistent between tests except where otherwise noted.

4.1 Generating Large Signature-Sets

An obvious issue with testing an IDS's scaling performance in respect to signature-set size is that there are simply not that many signatures available. Even if all of the pre-existing and community developed signatures were included, the signature-set size would still number under 20K signatures. An obvious solution to this (albeit one that is only useful for testing purposes) is to generate random signatures. It is desirable that the random signatures are statistically similar to the existing signature-set in respect to the string and regular expression features used. For the purposes of generating a large number of random signatures, a straightforward algorithm using non-parametric statistics was used to generate a signature-set of approximately 800K signatures.

First, existing signature-sets were split into individual features, each feature appended to a file by its label. These files contain as many duplicates and unique features as there are duplicates and unique features in the actual signature-set, totaling approximately 27,000 unique features. Second, based on the number of signatures desired, each signature in an actual Snort signature-set is used as a template which is permuted thousands of times by replacing each feature value with a sample drawn randomly from the file for the feature label. This was done in order to retain a similar distribution of the feature labels and total number of features within each signature. Lastly, invalid and duplicate signatures were removed by eliminating signatures which did not pass Snort's signature parser. In the resulting signature-set, each random signature has the distribution of feature values as the global signature-set and the same set of feature labels as its parent template.

One issue with this approach is that many invalid and duplicate signatures are generated and must be removed in order for the signature-set to be used. Of 1.2M signatures randomly generated, only 800K were remaining after removing duplicates. This leads to the random signatures being biased towards more complex signatures with a larger number of features.

Chapter 4. IDS Performance Characteristics

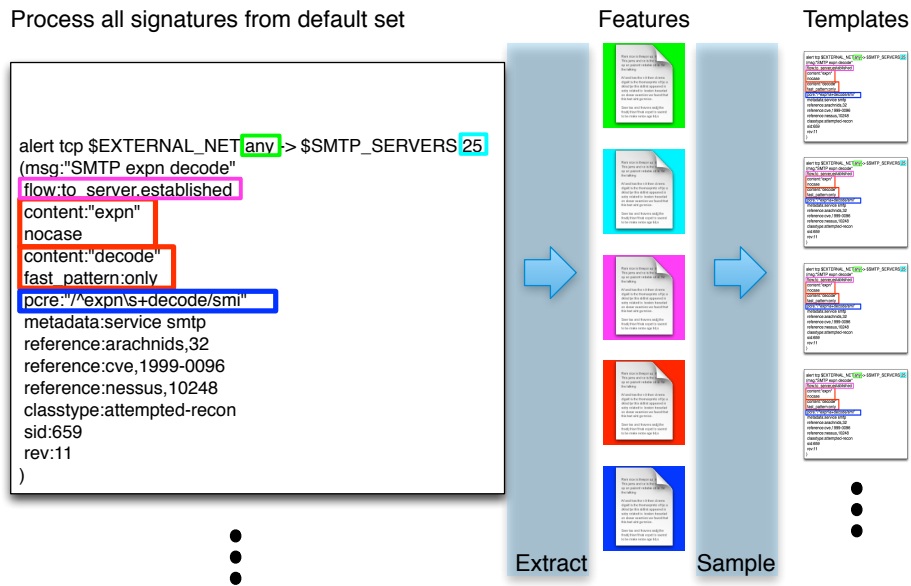


Figure 4.1: Random feature replacement algorithm

We can generate distributionally similar random signature-sets by randomly replacing features within existing signatures.

The average signature length in the Snort Vulnerability Research Team (VRT)¹ signature-set is 498 characters whereas the average signature length in the randomly generated signature-set is 638 characters, a difference of about 22%. This was expected and was shown to increase per-packet processing costs proportional to the increase in string length. This will result in an overestimate of Snort performance cost for all signature-sets sizes. Since all signature-set sizes incur this additional cost, the relative differences in performance will still provide a valid estimate of potential performance gain in a hypothetical anticipatory system.

¹<http://www.snort.org/vrt/> - Snort Ruleset for version 2.9.0.3 (File - Retrieved on Oct. 17, 2011)

4.2 Snort IDS Performance Scaling

4.2.1 Startup Performance

One of the first issues discovered in the current version of Snort is extremely slow startup times for large signature-set sizes. Startup performance is generally not relevant due to small signature-set sizes and long run-times. However, it can result in experiment run-time being dominated by startup time. As a result, experiment sizes were kept relatively small ($< 50K$ signatures) in respect to the signature-set available from the randomly generated set of $\sim 800K$ signatures (see Chapt. 4.1 on page 91). A functional model for startup time was also required so that experimental measurements could be delayed until the IDS was ready for input.

An experiment was run to determine startup time which would limit experiments. This experiment made use of the Linux “time” program while running Snort against a small file containing a small sample of 10 packets. The packet processing time in this case did not have a measurable impact to the results for the sizes of signature-sets tested. 10 trials were run at increments of 1,000 signatures for each signature-set size between 0 and 45,000 ($n = 450$). For the randomly generated signature-set, a 5th order polynomial provided a best-fit ($R^2 = 0.9998$) as shown in Fig. 4.2. A single test run of 10 trials using 100K signatures corresponded closely to the polynomial model. As a result, the current startup performance hinders the use of extremely large signature-sets for experiments with a large number of signatures. The poor startup performance is likely the result of assumptions made by Snort developers about the largest sizes of signature-sets that would ever be used. Indeed, most performance optimizations of Snort configurations include removing as many signatures as possible without introducing “too many” false negatives. Nonetheless, large signature-sets can still be used, but must be split between multiple Snort instances to achieve reasonable startup times. It is also possible that there are configuration options which may alleviate the issue, though these are not known.

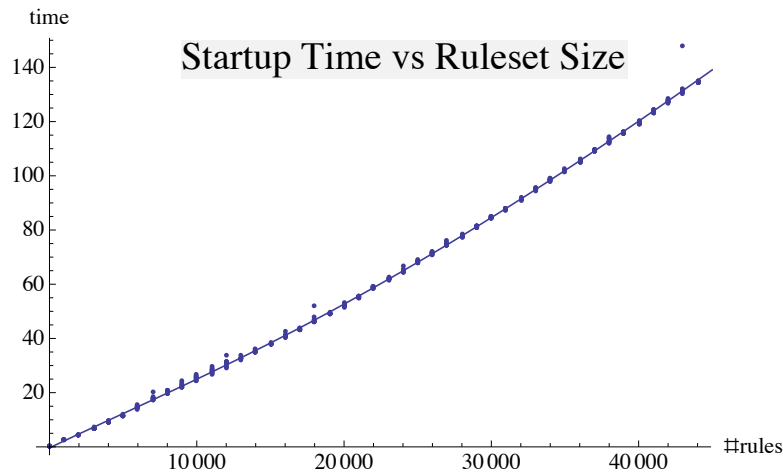


Figure 4.2: Polynomial fit for the startup time of the Snort IDS
 Tests with larger sized signature-sets suggests that the startup time is exponential, but the best fit with the experimental data is a polynomial. The best-fit has units of seconds:

$$-0.363 + 0.0026x - 1.918 \times 10^{-8} + x^2 + 1.652 \times 10^{-12} + x^3 - 3.109 \times 10^{-17}x^4.$$

4.2.2 Packet Coverage

The expected coverage of the Snort IDS when using a *default* signature-set was determined by running the IDS against a flat file containing 117GB of Packet Capture (PCAP) data. The data-set represents about 6 hours of traffic between the hours of 9am and 3pm from our corporate network. One caveat of the data-set used is that although collected outside of our corporate firewall, all HTTP and HTTP Secure (HTTPS) traffic was seen through a proxy server. This should not significantly affected the alert output as all individual TCP sessions can still be identified, although we did note Snort error messages regarding upper limits on session tracking. These and other limitations will generally produce an under-estimate of the coverage, so are not a confounding factor regarding measurements.

For coverage measurements, Snort was configured to use the entire official “VRT” signature-set as described in Appendix A.1. All detection signatures except those marked “DELETED” or which failed to compile were enabled in the configuration, representing approximately 4,300 signatures. All of Snort’s built-in preprocessors which were required for this signature-set were left enabled (i.e. stream5, ssl, rpc_decode, bo, dcerpc2, ssh, smtp,

Chapter 4. IDS Performance Characteristics

frag3, and http_inspect). Non-signature-based event generators (those with an generator Identifier (ID) > 1) were disabled or suppressed as it is the coverage of the main signature-based detection engine that was of interest.

Each packet or stream may produce multiple Snort alert events. To eliminate duplicates for purposes of measuring coverage, only events with a unique tuple consisting of (proto, sip, sport, dip, dport, timestamp) were counted. Additional events also occur when re-assembled TCP traffic is processed by a specialized processor. This was ignored, but is not expected to conflate event counts. Many individual packets would result in single events. In many cases if stream-based detection were not used, no event would be detected for any individual packet. As a result, stream-based events should result in an underestimate of A_p .

For a 117GB corporate dataset, the default Snort signature-set results in a packet coverage of $A_p = 0.04\%$. By our definition of *coverage* (Sect. 3.1.3, page 64), if every packet were to produce at least one alert the maximum usable proportion of packets would be 100%. The small coverage of the default signature-set represents a kind of upper bound on any gains via anticipatory bias. This is an upper limit on the proportion of packets which might be used to provide a bias on future detects. As most events are not useful for prediction, the A_q term is needed which describes the ratio of events which are useful for biasing the detector using a particular predictive engine. For some types of traffic and some attacks, each event may result in multiple predictions, each resulting in a small performance gain. The coverage ratio, A_r is thus an underestimate of the actual coverage (which is data dependent).

As mentioned in Sect. 3.1.3, underestimates in coverage will result in underestimate of performance gains. This is acceptable since we desire an estimate of the least amount of performance gain to expect.

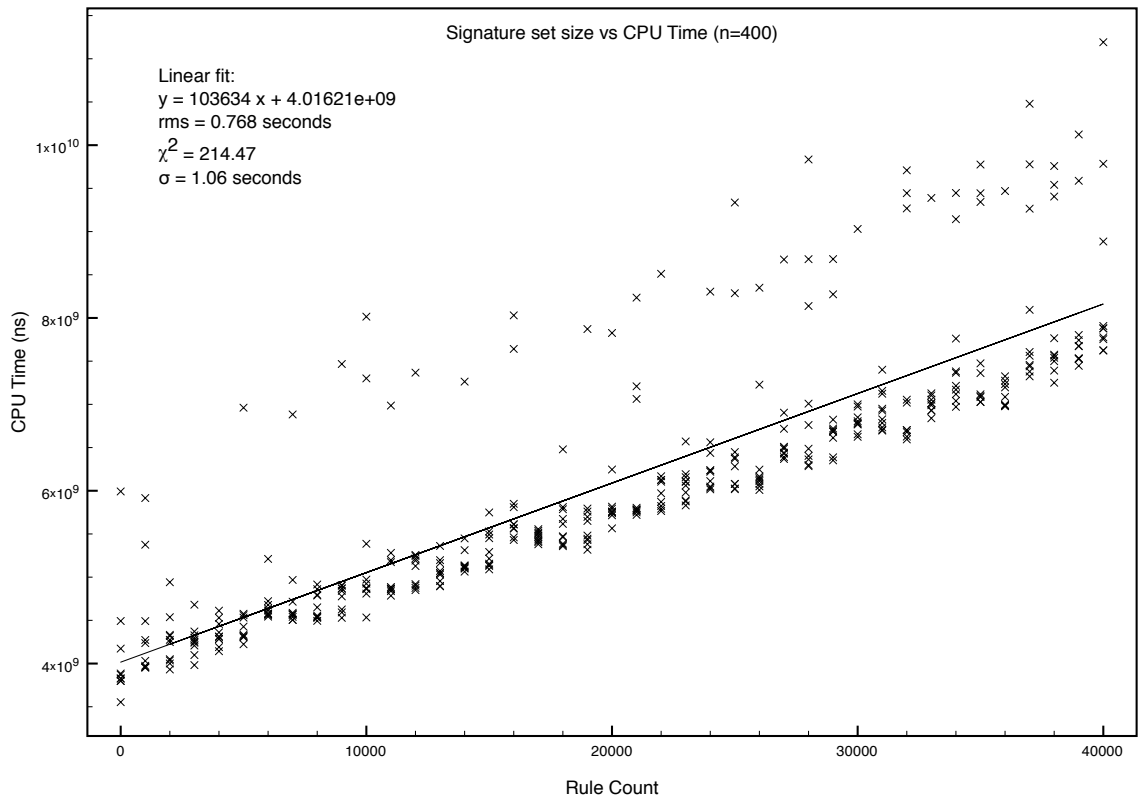


Figure 4.3: Linear scaling of total CPU-time with signature-set size of the Snort IDS

4.2.3 CPU-time scaling

Since we are comparing the cost of Snort instances running with various sized signature-sets, we can gauge the computing cost by measuring the performance on a test platform as signature-set size is increased and finding an acceptable function to model the system’s behavior as shown in Fig. 4.3.

A linear fit results in a simple estimated cost function in terms of nanoseconds of CPU-time:

$$C_{\text{time}}(x) = 103634x + 4.01621 \times 10^9. \quad (4.1)$$

It was expected that the scaling performance of the Snort IDS was linear in respect to signature-set size as the version of Snort being assessed uses an Aho-Corasick algorithm for string matching[33, 68]. This algorithm's complexity is linear in the sum of the length of patterns and string being matched, both for construction of the Deterministic Finite Automaton (DFA) and for pattern matching[3]. For Snort, linear scaling is essentially true, as shown in Fig. 4.3 with the glaring exceptions of packet drop rate and startup time as described in the next section.

4.2.4 Packet Loss Scaling Performance

Unfortunately, estimating performance by simply measuring total system CPU-time is eventually confounded by shared-resource issues when measuring performance of Snort running with large signature-sets. Large signature-sets, while incurring only a proportional increase in CPU-time, result in substantial packet loss on the front-end of the IDS. This finding is the reason that packet loss is included in the cost function for the system. Without considerations of packet loss we might only slightly overestimate performance costs for small signature-sets but we would grossly underestimate performance costs for larger signature-sets.

Several experiments were run to learn the packet drop rates as a function of signature-set size. The randomly generated signature-set was used to determine a worst case scaling rate. For these tests, the alerting and logging facilities of Snort were disabled. In this way Snort will not block due to I/O constraints on output alerting and logging costs. This is particularly relevant when measuring performance in respect to signature-set size as even small sets of randomly generated signatures are likely to produce a larger number of alerts than conventional signatures.

Figure 4.4 demonstrates that while signature-sets of size n increase the system overhead proportionally, the larger complexity of the signature-set increases packet drop rates by up

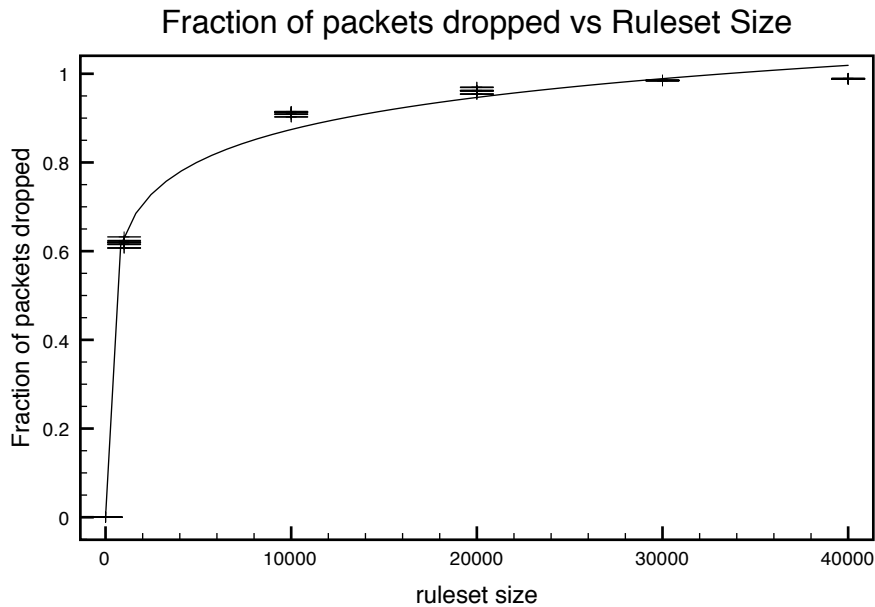


Figure 4.4: Test 18-2: Packets dropped by Snort for signature-set sizes 1K-40K
60 trials (10 per set size) were run at throughput of ~ 400 Mbps.

to a factor of 10. The cause for the high packet drop rate is likely that the Snort process is starved for CPU-time due to system limitations and processor contention due to I/O IRQs, possibly due to configuration issues such as a high new API (NAPI) *budget rate* as suggested in [81]. The unfortunate side effect is that the total number and type of alerts produced is not monotonically increasing as signature-set size increases (see Fig. 4.5). It is unknown if the set of alerts produced in cases of high packet loss is biased in any way, though this would be interesting to learn.

High packet-loss issues have been noted by others and various methods have been used to limit packet loss[81]. In our case, however, as the number of signatures is increased, contention for the CPU will eventually result in the same problem. As a result, the overall shape of the packet loss function is unlikely to change significantly, though this has not been thoroughly explored.

Chapter 4. IDS Performance Characteristics

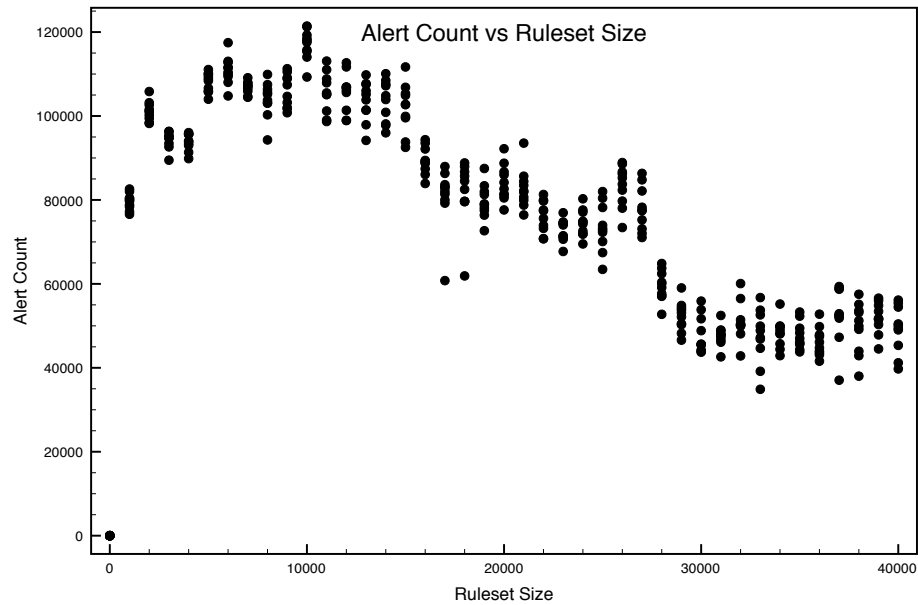


Figure 4.5: Test 18-1: Snort scaling for signature-set sizes from 1K-40K

10 trials at each of 1K through 40K signature-set sizes for a 1M packet sample on a live ethernet interface. Packets were replayed at the maximum interface speed (~ 400 Mbps). The non-monotonicity is due to high packet drop rates. Coverage measurements must therefore be based on cached PCAP data to eliminate drops from I/O blocking and CPU contention.

If we measure Snort's packet processing rate using the built-in performance monitor we can estimate the performance bottleneck as an exponential function. On the test system, an acceptable functional model was found:

$$PacketProcessingRate \leq 1.0392^{-0.008x} + 0.0583 . \quad (4.2)$$

As can be seen from Fig. 4.6, this fit is a gross over-estimation of experimental measurements, but sufficient for our purposes. An over-estimation of the number of packets successfully processed will result in an under-estimate of the effects of increasing the number of signatures in a particular IDS instance. For the test system the packet-loss rate is:

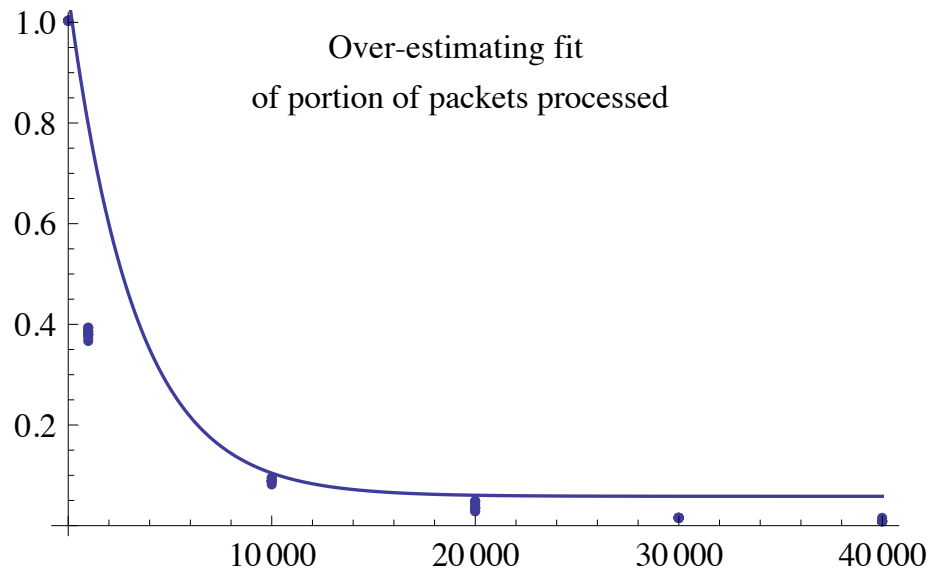


Figure 4.6: Over-estimating fit for packet processing rate

$$C_{\text{loss}}(x) \geq 1 - (1.0392^{-0.008x} + 0.0583)$$

$$C_{\text{loss}}(x) \geq 0.9417 - 1.0392^{-0.008x} . \tag{4.3}$$

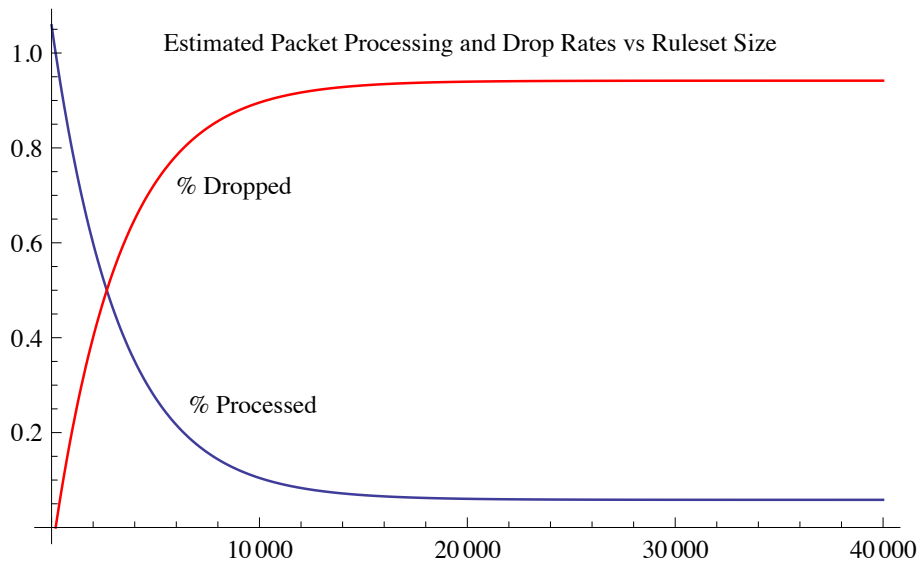


Figure 4.7: Packet processing rate estimate curves

Chapter 4. IDS Performance Characteristics

It is important to reiterate that other architectures, operating system kernels, and configurations will result in different scaling performance. For example, the scaling performance on a Macbook Intel i7 running the same test resulted in only slightly differing constant terms. Further, if multiple processors were in use by the IDS (either by splitting the signature-set or splitting the traffic across multiple IDS instances) the scaling performance curves should retain the same basic shape.

4.2.5 Anticipatory Gain Potential of the Snort IDS

For the experimentally derived cost functions in the previous section, we can explore the parameter space to get an intuition on when (and why) prediction is a useful tool for performance improvements in terms of performance gain. We can use the experimentally derived cost functions and our anticipatory gain model to derive a set of curves which are representative of average gains for a hypothetical anticipatory system. Deriving such curves is straightforward so long as closed-form estimates are available for the cost functions. Appendix D.2 gives the Mathematica script used to calculate the gain curves shown.

The packet coverage, A_p of one of Snort's VRT signature-sets test is only a fraction of a percent. For example (ignoring thresholding issues, only considering detection signatures, and ignoring IPv6 traffic), a 117GB sample of corporate traffic generated 70,000 alerts from the approximately 186 million packet events, making $A_p \approx 0.038\%$. If this number seems low, note that only the detection engine is producing events. All other preprocessors are disabled or suppressed. A current prototype predictive system has an overhead cost of 20% (roughly estimated from experiments). This system's coverage is so low that even a $k = 1$ value and a perfect predictor for all detected events (i.e. $A_q = 1$) would be far from covering overhead costs, gaining only 0.3% in CPU-time per packet costs. It is clear that *much* higher coverage is necessary for appreciable performance gains.

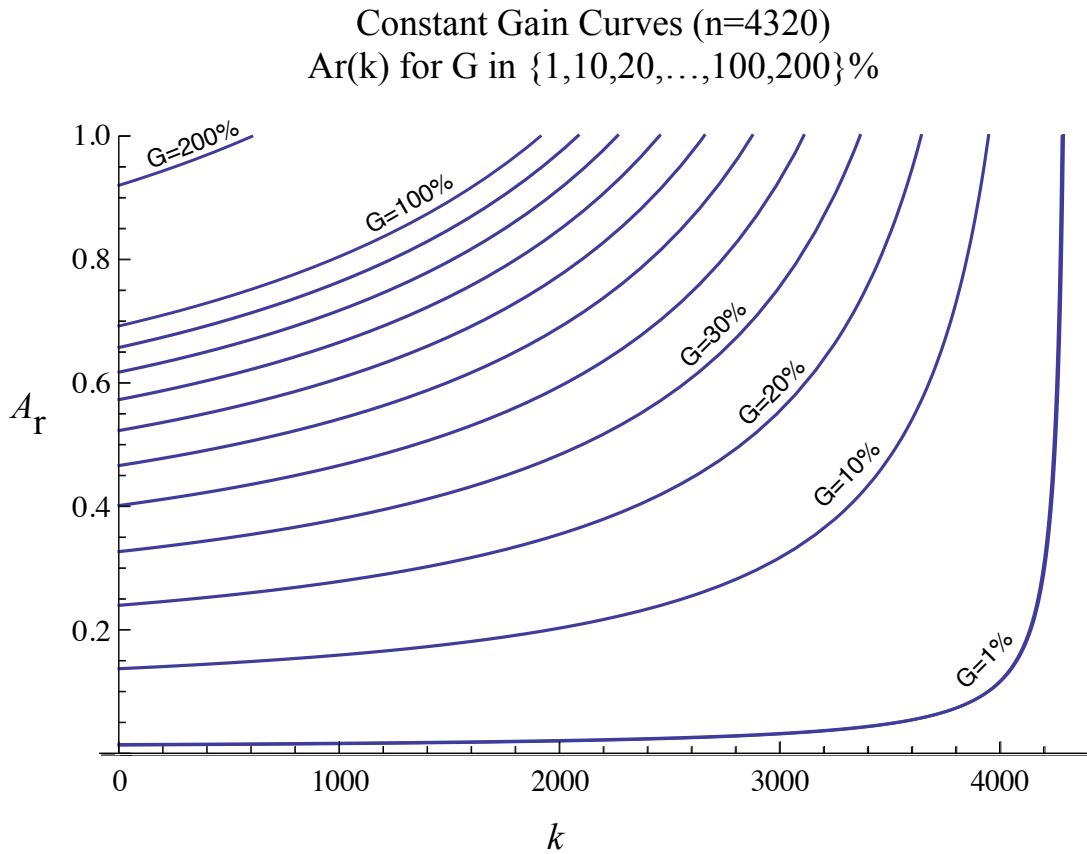


Figure 4.8: Curves of constant gain for signature-set size of $n = 4320$

Family of curves was derived by solving the gain function for the coverage ratio, A_r , in terms of the size, k , of a single secondary signature-set.

If we turn the problem around, for the default signature-set, with $k = 50$ (reasonable given the power-law distribution of event types), a net performance gain above a 20% overhead requires $A_r \geq 21.7\%$. In order for anticipatory methods to be applicable for performance improvement of the existing Snort IDS, predictor coverage (and thus packet coverage) would still need to be dramatically increased. Indeed, tests where a dataset and signature-set are artificially biased to have a high coverage of incoming data demonstrate the potential of a prototype anticipatory system (see Chapt. 6.2 on page 119).

It is important to note that while increasing coverage does not significantly increase basic (non-anticipatory) detection cost, it can significantly increase I/O costs. The best

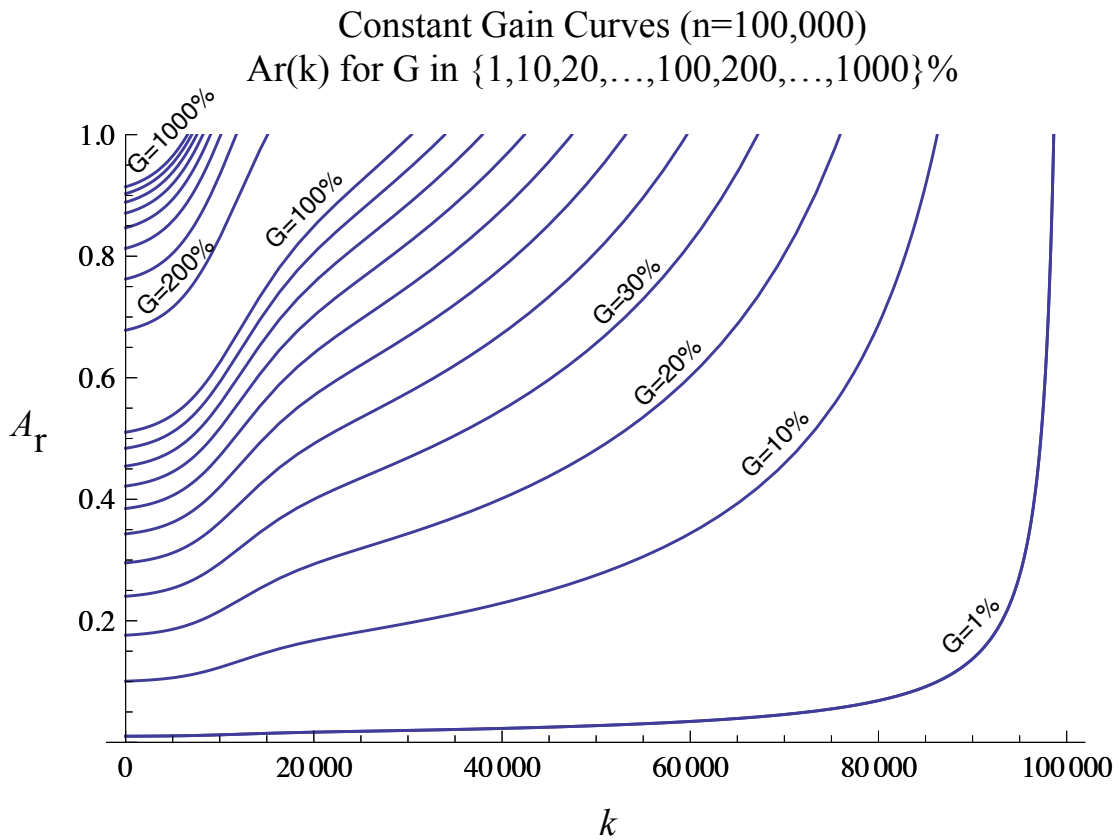


Figure 4.9: Curves of constant gain for signature-set size of $n = 100000$

Family of curves was derived by solving the gain function for the coverage ratio, A_r , in terms of the size, k , of a single secondary signature-set.

performance gains can be achieved when a small number of additional signatures cover a large portion of the traffic. In these cases the total number of output events needs to be optimized to prevent wasted work (see Chapt. 3.1.1 on page 58). For purposes of a simple anticipatory gain analysis we can temporarily ignore issues with I/O costs of high-coverage signature-sets and assume that these can be alleviated using clever signature activation policies (such as those described in Chapt.3.2 on page 70). Figures 4.8 and 4.9 demonstrate performance scaling in respect to IDS coverage, showing constant gain curves in terms of the secondary signature-set size, k , and the predictor coverage ratio, A_r . For large signature-sets, the number of anticipatory signatures needs to be a small fraction of the total signature-set size for appreciable gains in performance. Interestingly, if the number of

Chapter 4. IDS Performance Characteristics

anticipatory signatures is small the performance gain increases non-linearly as coverage is increased. While this is partially an artifact of the analysis (assumes $1 - A_p$ packets are wholly processed by the smaller IDS instance(s)), it is clear that there are acceptable trade-offs to be made and significant performance gains to be achieved.

Chapter 5

Prototypes

5.1 Packet Wrangler

The Packet Wrangler prototype was designed to perform all prediction and packet forwarding external to the IDS. This was intended to allow swapping of the IDS and the predictor. It is possible to use Packet Wrangler, without significant modification on fundamentally different IDS engines (e.g. BroIDS, Snort, etc.). The approach is summarized in Figures 5.1. The background of approach is discussed in Chapt 2 on page 56.

In the prototype, each Snort instance is attached to and sniffing packets from a TUN/TAP virtual device rather than a physical interface.¹ The prototype uses the open-source “PF_RING” (*packet-filter* ring) kernel module developed by the Luca Deri of the Network Top (NTOP) group.² This kernel module was experimental when this research began, but is being considered for inclusion within mainline kernel distributions. In general, it provides dynamic packet filtering and forwarding mechanisms with as few as possible

¹A virtual device approach was chosen for simplicity of implementation although other methods were possible.

²<http://http://www.ntop.org> - *NTOP Homepage* (PF_RING Source Code - Retrieved on Jan. 17, 2012)

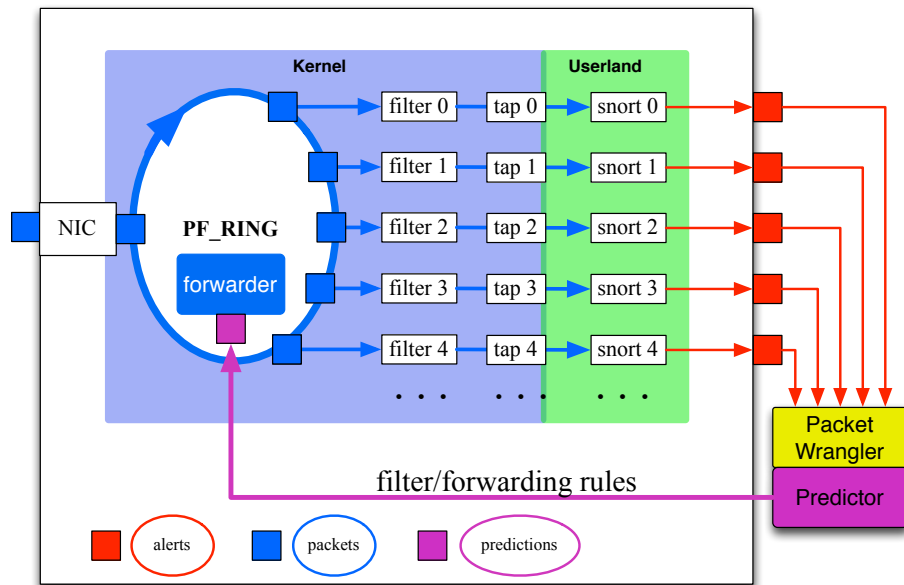


Figure 5.1: Predictive IDS architecture showing data flow

The data flow describes the interaction of software components when processing packets, alerts, and predictions.

memory copies[32]. This, in essence allows each Snort instance to read independently from the same kernel ring-buffer with minimal duplication of data.

The prototype communicates internally via a combination of Unix Sockets (*snort socket server*), Inter-process Communication (IPC) Message Queues (*forwarder*), standard I/O and Application Programming Interface (API) calls (*Packet Wrangler*), and direct device reads (*Snort*). Additional, Packet Wrangler listens on a Unix Domain Socket for command messages for monitoring and changing internal state. All software except for Snort itself are custom components and programs.

5.1.1 Startup

The system is started using a set of shell scripts which results in the following sequence:

1. Create and set device options for virtual devices, on per equivalence class of signatures.

Chapter 5. Prototypes

2. Start Snort instances *snort0* through *snortN* (*s0...sN*), on per virtual device, each configured with a different signature set.

Input: devices *tap0* through *tapN*

Output: Snort Unix Socket

3. Start the Unix Socket server for event/alert processing.

Input: Snort Unix Socket

Output: stdout1

4. Start the Packet Wrangler program.

Input: stdout1

Output: IPC Message Queue 1

5. Start the PF_RING “Forwarder”.

Input: IPC Message Queue 1

Output: pfring filter states via API calls

The system initially starts with all packets being seen only by Snort instance *s0*, which in most experiments was configured with a large default signature-set. Snort is configured to send all alert events to a Unix Socket. When there are multiple instances the alerts read from the socket are interleaved. Each event ID is tagged by the Snort instance. The Snort Socket Server reads events and forwards them via standard output to the Packet wrangler program. Packet Wrangler processes each event seen on its standard input, making predictions, and sending messages to the Forwarder to update the PF_RING forwarding rules. Each forwarding rule added to PF_RING represents a single connection and new virtual device destination. Until the rule is removed, all future packets for the connection are forwarded to the virtual device specified by the forwarding rule. It is important to reiterate that packets are read only once.

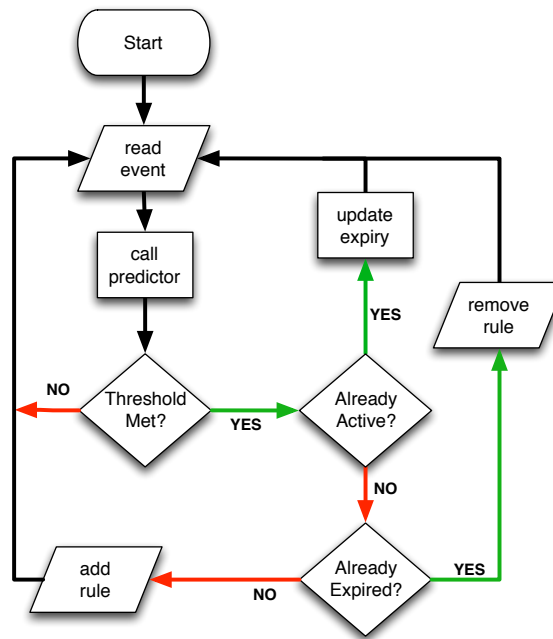


Figure 5.2: Flow diagram for the Packet Wrangler program

5.1.2 Program Logic

Packet Wrangler has a fairly simple internal logic. This is summarized by the flow diagram in Fig. 5.2. Upon receiving an input alert event, it first determines whether a prediction can be made. If so, it first checks whether there a previous prediction is still active. If so, it updates the expiry time for the old prediction. If not, it checks whether the prediction meets configured confidence thresholds. If the confidence thresholds are met and the expiry time has not already passed, then a message is sent via IPC to the *Forwarder*. The Forwarder reads each IPC message and asserts a new packet forwarding rule with the specified connection tuple. The Forwarder reads the message queue in a blocking mode to ensure that messages are read as soon as they arrive. Rule expiry is performed opportunistically on every loop by checking a priority queue and sending “remove” messages to the Forwarder.

5.1.3 Predictors

Any anticipatory approach must address two primary questions: a) which correlations can be learned and leveraged; and b) what risks are acceptable in respect to additional detector errors. While there is no *right* way of predicting future events in all cases, there are several candidates which have been explored, primarily: rule-based and Naive Bayes predictors. Although these techniques have been applied independently it should be clear that multiple methods can be combined.

Rule-Based

A rule-based predictor applies formal logic to predicting future outcomes. In the simplest case, an If-Then rule is constructed for which the premise is some state of a network connection or event produced by the IDS and consequent is a likely future state or event. Rules can be probabilistic, with assigned confidence measures being used in a Maximum-Likelihood manner or in a non-deterministic manner. Chaining of signatures may also be used to propose future states based on complex sets of priors.

Both rule-based and Bayesian predictors are generally easy to analyze. However, rule-based predictors require expert knowledge on what predictions can be made and confidence estimates either imposed arbitrarily or learned from experimental data. Provided expert knowledge is available, rule-based predictors are also the most trivial to implement. Trivial If-Then rules may be constructed by either applying expert-level domain knowledge or deriving rules from observed correlations. More complex rules can also be constructed, but require deeper insight into network and attacker activities.

A rule-based predictor implemented as a stochastic predictor is essentially just a Bayesian predictor with a fixed set of hand-picked priors. One issue with this approach is

Chapter 5. Prototypes

that signature-based predictors are not readily amenable to online-learning beyond learning simple confidence measures.

For the Packet Wrangler prototype, a set of new IDS signatures were constructed to both expand packet coverage and to implement a rule-based predictor. The specific signatures used are given in App. B.3 on page 170. If an event is seen by Packet Wrangler which meets the preconditions for one or more predictor rule, the rule is applied and new forwarding rules are added to the Forwarder.

Naive Bayes

A Naive Bayes approach is simple to construct, but requires a large amount of representative training data. Naive Bayes also provides a simple and computationally inexpensive mechanism for learning and prediction. One of the issues with the Packet Wrangler approach is that prediction must be made very quickly. Any delay and the value of a prediction may disappear. Most predictable event sequences occur within the first few seconds after the initial event. As such, a fast predictor was needed. The current approach is limited by dataset size (resulting in generalization issues), distribution assumptions, non-stationarity issues, and issues with learning window-size.

The current Bayesian approach used within Packet Wrangler prototype uses offline learning. After learning sequence statistics, the predictor represents the probabilities of all event sequences for the training set. Although spatial correlations between connections could be learned, the approach has not been applied due to the computing expense (n^2 comparisons for n connections each time window).

The basic operation of Packet Wrangler is identical with either the Bayesian or the Rule-Based predictor.

5.1.4 Discussion

The intent of the prototype is to treat each each network connection distinctly in terms of applicable signature-sets. When possible, we want to forward all packets for individual connections to IDS instances with a smaller signature-set.

Clearly, the prototype will incur additional overhead costs. When there are many events output from the IDS, this cost can be substantial. The approach also incurs a nominal increase in memory requirements. Each secondary IDS instance duplicates the overhead and a portion of the decision procedure of the primary (unbiased) IDS instance.

Costs notwithstanding, with a sufficiently large signature-set and sufficiently compact secondary signatures sets, the Packet Wrangler approach is capable of improving performance in terms of overhead costs. As shown in Chapt. 6, the decrease in overhead comes at the cost of incurring additional errors.

An external anticipatory model as exemplified by the Packet Wrangler prototype may be useful for a number of reasons:

- The approach models anticipation using any choice of predictor.
- Alternate connection tuples can be used such as an IP 3-tuple (source IP, destination IP, protocol).
- Predictions can be dynamically calculated or learned based on current traffic.
- Predictions can be made which concern distributions of events for various subsets of computing machines, traffic, software, etc.
- Multiple IDS implementations can be used within secondary instances which differ in abilities and specialization (e.g. an HTTP-only IDS).

Chapter 5. Prototypes

Nonetheless, there are a number of valid criticisms of the approach. First and foremost is the complexity of managing traffic flows external to an IDS. Many IDS have internal traffic management and tracking mechanisms. Second, the total number of signatures available and the threat coverage of existing signature-sets is quite small. As discussed in section 4.2.2 the average coverage of a recent default Snort signature-set is a fraction of a percent. As such, the model is limited without careful addition of high-coverage signatures.

Flow Tracking

For someone familiar with the design of the Snort IDS, one obvious issue with the approach is loss of session tracking information within the secondary instances. When packets are redirected to a secondary IDS instance, in almost all cases the initial three-way handshake will have never been seen by the secondary instance. This completely disables the flow tracking abilities for signatures within a secondary IDS instance. Unfortunately, most signatures rely on the state of a TCP stream. Within the current prototype, any signature within a secondary IDS will not know whether the packets received belong to an established connection.

While this is problematic, it can be partially dealt with by using paired sets of secondary IDS instances. For each possible tracked session state we partition the secondary signature-set (for Snort this is either “to_server/from_client” or “to_client/from_server”). We then remove the flow tracking from all secondary signature-sets. Since each equivalence-class detection signature indicates whether a connection is established and the server-to-client direction we can infer the class to which future packets belong and forward them appropriately for the duration of the TCP session. The only caveat here is that occasionally negative conditional dependencies occur within signature options. This can result in additional false-positives when stream state is unknown.

Another possible criticism is that Snort's flow-tracking mechanisms (i.e. "Flowbits") already provide a straight-forward method of hiding complex signatures and enabling them only after some information is known about the connection. In other words, a signature-set optimization of TCP and User Datagram Protocol (UDP) flows (defined by a connection 5-tuple) is already implemented and utilized within the Snort IDS. A superior implementation for strict signature-set optimization would simply use the Flowbits and add new signatures for protocol and predictive feature identification. Indeed, this is essentially what is done within current Snort signature-sets for Secure Sockets Layer (SSL), HTTP, and other easily identified application protocols. One limitation of the built-in Flowbits approach is that the signature-chaining is fixed for the lifetime of the signature-set. Further, The Flowbits mechanism also precludes forwarding packets to an external IDS, desirable in our case as we would like to test systems which implement various IDS with fundamentally different capabilities.

5.2 Probabilistic Flowbits

As has been mentioned in previous chapters, the signature-chaining mechanism of Snort is entirely non-probabilistic. As shown in Chapt. 3.2, this results in higher error rates than a system which utilized probabilistic signature chaining. Probabilistic flow tracking allows us to achieve better trade-offs between detection errors and IDS coverage.

A straightforward method for implementing the probabilistic signature activation methods described in Chapt. 3.2 combines flow-tracking and signature-chaining to activate a subset of signatures according to a pre-configured set of probabilities. The end result is a per-flow biased sampling of high-cost and high-occurrence signatures with a fixed threshold. Future versions could easily use online learning and an adaptive threshold based on system load and packet loss. This follows prior work on adaptive IDS reconfiguration, but differs in that signatures are individually optimized on a per-flow basis. Prior work on adaptive

Chapter 5. Prototypes

reconfiguration of IDS has generally focused on heavy-handed tuning in which signatures are entirely removed from the signature-set[35, 92].

We can justifiably call the prototype “Probabilistic Flowbits” as it directly extends the “Flowbits” signature chaining mechanism of the Snort IDS. The modified Flowbit module allows both setting and checking of Flowbits with a probability which is specified within “Flowbit” option of a Snort signature. To keep the modifications to the existing source code minimal our current implementation uses a fixed set of discrete probabilities which can be selected by using newly added Flowbit operations. For example, to set a Flowbit named “**bit1234**” with probability 0.1 we would use the Flowbit operation “**isset010**”. The signature option within a Snort signature would be written as:

```
Flowbit:isset010,bit1234;
```

Similarly, using the operation “**set010**” would set the specified Flowbit with probability 0.1.

Probabilistic Flowbits have been implemented differently than what might be obvious from the standard Flowbit mechanism. In both the standard mechanism and the modified implementation, when the “**set**” operation is used, the Flowbit will eventually be set for the remainder of the TCP stream. However, the “**isset**” operation actually ignores the internally stored Flowbit. This operation simply returns true with the chosen probability irrespective of how many times the bit is checked or whether an actual bit-check would have returned true. This not only allows us to control signature activations by applying the standard signature chaining mechanism probabilistically (e.g. α_1 activates α_2 with probability $P(\alpha_1 \rightarrow \alpha_2)$), but also allows us to probabilistically activate signatures independently of any other signature firing an event.

Due to the simplicity of the implementation, minimal modifications were necessary to the existing source code of the open-source Snort IDS. This minimally modified system was sufficient to test the potential of probabilistic signature activation policies. A more

Chapter 5. Prototypes

sophisticated prototype is currently being constructed which uses online learning of alert statistics and dynamic thresholds. A discussion on potential extension to our prototype is discussed in the conclusions of this dissertation.

The modified Snort binary results in a small increase in signature calculation costs for those signatures that use the new keywords. The additional cost is due to larger numeric comparisons. Rather than single bit tests as are performed for standard Flowbit tests, numeric comparisons are performed between integer sizes. These have a worst-case cost increase of a small constant factor related to the integer precision used for comparisons. Nonetheless, this additional cost is not incurred for all signatures (only those that are being probabilistically activated). The gain achieved by removing signature computation costs generally outweighs the small additional overhead.

In an analogical way, the signature-activation policies are loosely similar to habituation (*relevance policy*) and desensitization (*cost policy*) within living organisms. In our analogy, each packet is akin to an input stimuli, each signature and its associated computing costs is akin to some perceived effect on the organism, and an alert is akin to a precept.

In the relevance-policy, when a signature fires frequently it is enabled rarely, noisy alerts being easy to spot and multiple detections on the same flow being somewhat superfluous. If the probabilities are correct for the input data, we have ignored input stimuli but not decreased false negatives. For our biological metaphor, a flood of identical input stimuli results in ignoring a large number of them (diminished response to a stimuli). If the number of such stimuli were to decrease substantially we would expect to incur additional false negatives.

For the cost-policy, we are addressing the fact that painfully expensive signatures can account for a large portion of the overall computing cost. The IDS is sensitive to these signatures being active for a significant portion of the input data. By enabling the signatures with probability inversely proportional to their cost, we incur less of a penalty in false

Chapter 5. Prototypes

negatives by making resources available for less expensive signatures. The system becomes less sensitive to the high cost signatures as their activation probabilities is decreased. In an organism, if an input stimuli cause some perceived negative effect, desensitization would decrease the negative effect of each perceived stimuli.

Chapter 6

Experiments

6.1 Methods

Both the hardware and software environments were kept relatively stable throughout most of the experimental configurations and various implementations. For detailed information about the hardware and software used, see App. A.1 on page 156.

To control for potential confounding factors several steps were taken. Software versions and signature-sets were kept consistent for each series of experiments. Problematic software services were disabled for both machines during experimentation (e.g. indexing services, backup services, etc). All experiments were also tested in both Exp-Control and Control-Exp sequence to ensure that experiment order had no effect (e.g. residual processes, memory leaks, etc).

For system measurements, the open-source “SystemTap” monitoring software was used for all process monitoring on the test system.[36] A customized SystemTap script was used to track cumulative CPU-time for all processes on the system with a sampling period of 10 seconds. This script is simply a modified version of the “schedtimes.stp” available with

Chapter 6. Experiments

the SystemTap software modified to periodically output cumulative statistics¹. Process monitoring was performed for the duration of the experiment with 60 seconds allowed for startup and process completion, prior to and post transmission of network data respectively.

Within experiments both the control and experimental setups use the same definitions for false negative and false positive errors. Although both the absolute and normalized error rates were measured, the normalized error rates are reported. An event is considered a false negative if we do not detect at least one of each event which is present in each stream. An event is considered a false positive if it should not have been detected but but any number of detections occurred.

Ground-truth event data was generated by running the IDS directly against a packet capture file with the configured signature-set (with no probabilistic activations or anticipation mechanism). As such, ground-truth event and profiling results represent the IDS running in an *offline* mode where it reads directly from a flat-file containing the same data which is sent over a network connection. The offline mode of the IDS guarantees that each packet will be processed by the IDS. However some characteristics of how traffic is sent and received results in some events which are not detected in the ground-truth configuration which are detected in control configurations (.e.g dropping of a portion of corrupt packets when transmitted, lack of packet loss at network interfaces or in-kernel, etc.).

Error rates for both control and experimental setups were measured relative to the file-based ground-truth and to each other. Increases in false positives are seen due to a combination of dropped packets and negative conditional dependencies between signatures as discussed in Sect. 2.4 on page 37

The datasets used within experiments are described in detail in App. A.2 on page 158. In brief, most experiments were run from samples taken from a 117GB file containing captured packet data from a corporate network. The sole caveat of the dataset (relevant for

¹<http://sourceware.org/systemtap/examples/process/schedtimes.stp> - *SystemTap SchedTimes.stp* (Script - Retrieved Aug 12, 2012)

some tests), is that having been captured from outside of a firewall and proxy server, all port 80 traffic is tunneled through the proxy making it more difficult to discern individual HTTP connections. This makes some experiments more difficult to perform, but is not a confounding factor for the results given. For several tests the dataset was partitioned into smaller segments for easier processing and for testing against adjacent time windows. The entire dataset represents approximately 6 hours and as such precludes exploration of any diurnal effects.

6.2 Packet Wrangler

The Packet Wrangler prototype and experiments run the IDS as an end-to-end system across a small test network. The control configurations run a single Snort instance (labelled the “ s_0 ” instance) consisting of all signatures for the test configuration. Experimental configurations consist of an s_0 instance as well as a collection of additional Snort instances (s_1, \dots, s_N), each independently processing independent and distinct sets of forwarded packets using subsets of the available signature-sets (each representing an equivalence class of signatures).

For each experimental run, packets were initially directed to the s_0 instance which was configured with the entire default signature-set. When an event was seen at the output of the IDS it was looked up to determine a set of maximum likelihood sequences. If a confidence threshold was met, then the packets for the connection tuple of the alert were then directed to the IDS instances configured for the alert’s equivalence class (instances $s_1 \dots s_N$).

The original concept was that alert predictors would be clustered to identify various equivalence classes of events. However, a series of experiments were first run to determine the maximum possible performance gains using the existing signature-set without modifi-

Chapter 6. Experiments

cation (ignoring error rates). Although some work was done on clustering methods, this became secondary to demonstrating performance characteristics.

The initial goal was to determine whether sufficient data could be forwarded to sufficiently small secondary signature-sets to get a decrease in total system overhead. For each test the system was configured with a standard signature-set and several pre-defined signature-sets, one per equivalence class. A new Snort instance was configured for each additional equivalence class.

In most tests, a “connection” was defined as the 5-tuple over the IP address pair, port pair, and protocol. In later tests, other connection tuples were assessed (such as the 3-tuple of IP address pair and protocol). Unfortunately, some difficulties arose in the packet forwarding kernel module (PF_RING) being used. In particular, the full 5-tuple uses hash-based filtering and forwarding whereas any other set of features uses a wild-card filtering approach. Wild-card filtering only performed well when there were a small number of rules. This issue is probably relatively easy to address by modifying the PF_RING kernel module. Unfortunately, offline learning using 3-tuple connections resulted in predictors that still lacked the requisite coverage for substantial performance gains for existing signature sets.

A listing of the Snort configuration options used for the experiments are in App. B on page 164. In brief, any preprocessor which was necessary for a detection signature (generator ID 1) was enabled with default options. This included “stream5”, “http_inspect”, and several others. Dynamic preprocessors and binary signatures were disabled. For a full listing of Snort command-line and configuration options see the Appendix.

6.2.1 Naive Bayes Predictor

The first experiments used a Naive Bayes predictor trained on event sequence statistics gathered from ground-truth data and a standard Snort signature-set. The predictor in each test was a stochastic matrix representing the probabilities of alert sequences which were

Chapter 6. Experiments

actually present in the test data (see App. A.2.1 on page 160 for an example stochastic matrix based on the DARPA 1998 dataset). A fixed threshold was used for each experiment along with a fixed timeout, allowing packets to be re-directed back to the primary Snort *s0* instance after the timeout. Although larger secondary signature-sets were used, only results for a “Top 50” configuration are shown in which the signatures for the highest occurring alerts were used.

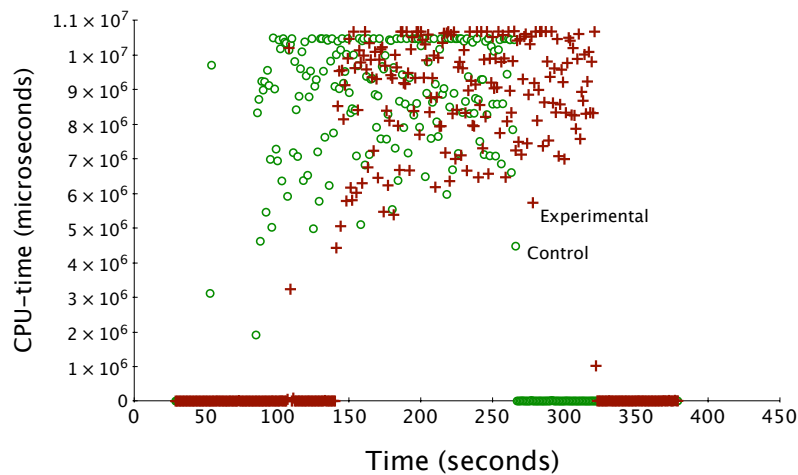


Figure 6.1: Test 5: Packet Wrangler forwarding using the *Top 50* event sequences

The time offset seen in the chart is due to a mistake made in scripting the start times for the experiment. This is inconsequential as it is the overhead times which we are interested in. The total duration of the experiment is dependent on the rate at which packets are transmitted over the network and is the same in both configurations.

Figure 6.1 is representative of most of the tests performed using the Naive Bayes predictor and any number of different configurations. In Test 5 a 17GB subset of a corporate packet dataset was used. Out of 85 different unique events the top 50 events accounted for 92% of events produced by the IDS. Obviously, when grouped into the same equivalence class these events also showed excellent correlation with one another. If we somehow had knowledge of which events would account for most of the incoming data, then we could forward these events to a smaller *s1* instance containing only the 50 signatures for the equivalence class (instead of the 4320 signatures present in the *s0* instance for this test). However, the experimental configuration generally performed worse than the control. The performance difference was caused by a combination of nearly identical primary *s0* IDS

Chapter 6. Experiments

costs between the experimental and control configurations and the added predictor costs associated with processing and prediction of events.

To confirm that the negative result was not simply due to a biased dataset, a larger test was run using the entire 100GB dataset. In these tests the dataset was streamed continuously and the Packet Wrangler program was remotely switched between control and experimental modes. For all of these tests run, there was either no significant difference or a decrease in performance in the experimental mode (Tests 7 & 8 [not shown]). The lack of performance gains even though a much smaller signature-set was being used needed explored. This led to the first portion of the assessment of Snort's performance scaling in respect to signature-set size as described in Chapt. 4 on page 90. It also led to a more thorough analysis of performance gains in respect to equivalence class sizes as shown in Sect. 4.2.5 on page 101

One potential issue with the configuration in early tests was that it was focused solely on decreasing system overhead. Therefore the incoming data rates were performed at the maximum speed of the outgoing network interface (~400Mbps). While this was identical for both the control and experimental configurations, high throughput would have minimized any gains which were measurable since the system would potentially be overloaded at all times and error rates were not yet being measured. In addition to identifying potential system load issues, the critical issue with the use of existing signature-sets was learned to be *packet coverage*.

If the top 50 events, when predicted, do not account for a significant portion of the input data, then few packets will be forwarded and no gains can be achieved. Since the stochastic matrix mostly tends to predict that the same event will occur multiple times in sequence, we will tend to forward packets when we see such an event. For the corporate dataset being used, the top 50 events accounted for the largest portion of total events, but only 0.012% of packet input. If the average number of packets in sessions containing these events were known, we might estimate the total packets forwarded, but with such a low coverage, this seemed pointless. Clearly, unless the packet coverage is significantly high, no gains are

Chapter 6. Experiments

possible. Interestingly, the training set has a packet coverage of around 20%, but only 4% is associated with the top 50 equivalence class calculated in the same way as for the corporate dataset.

The primary result in the first dozen experiments was simply that not enough information was being learned from the incoming data. While each packet was running the gamut of the detection engine, very few were producing any information. Even in the case of the dataset, predictable events accounted for only 4% of the incoming packet data, which was generally not enough to pay the costs of additional overhead of an external predictive optimization approach. During experimentation, the runtime scripts, Packet Wrangler software, and packet forwarding module were all improved after new issues were discovered in respect to edge cases and bugs in various modules (and those of product maintainers (Tests 9-14 [not shown])).

Test 15 was able to show a performance improvement for the dataset with the Top 50 configuration by reducing the data rate to 20Kpps (\sim 29Mbps). The performance gain for the DARPA dataset in the Top 50 configuration is shown in Figure 6.2. While this results suggests that the basic approach can gainfully shunt load to smaller IDS instances, the gains are only moderate and are limited to the DARPA dataset, which is not in any way representative of real networks.

Although the result is limited, what is particularly interesting is the portion of packets which are actually shunted to the secondary instance. While at first glance one might assume that a 13% gain indicates that some substantial portion of the 20% of event-associated traffic was shunted, this is not the case. Recall that at most 4% of the dataset can be gainfully predicted in the Top 50 configuration. For Test 15, control s_0 processed 13,576,440 packets and the experimental s_0 processed 13,252,939, meaning that in the experimental only 2.4% of packets were forwarded. This means that 2.4% of the packet data accounted for significantly more than 13% of the costs in s_0 . The 13% gain is the result of differences in cost of processing the forwarded packets.

Chapter 6. Experiments

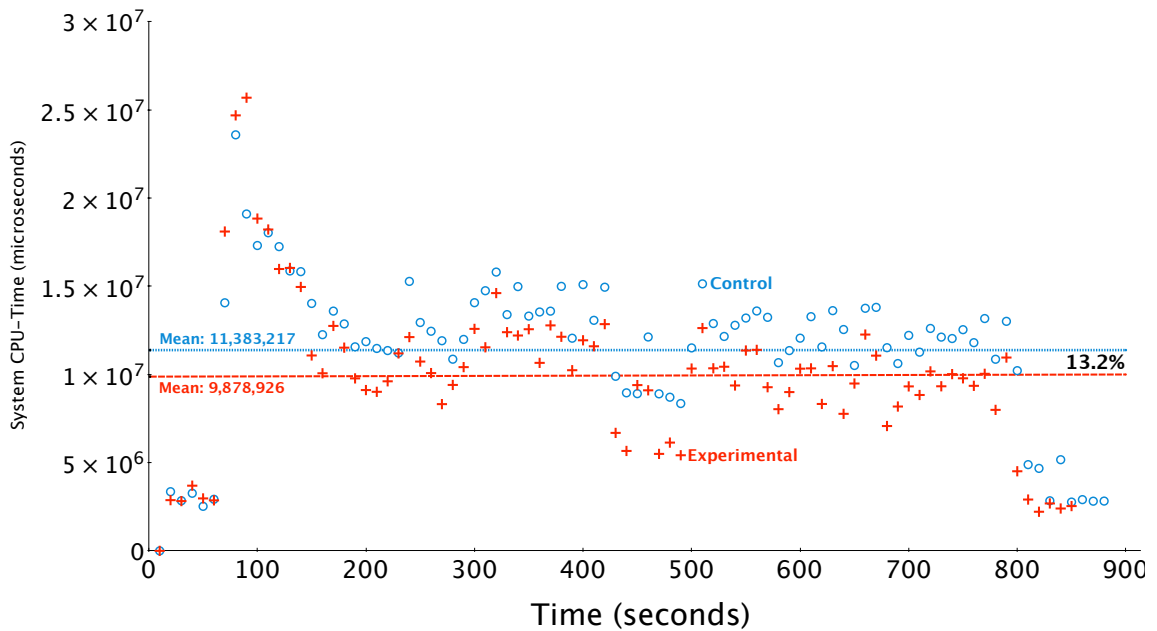


Figure 6.2: Test 15: Packet Wrangler Top 50 Configuration
Test uses the 1998 Dataset at 20Kpps.

It is important to reiterate that error rates were not being considered for these tests. Unless significant decreases in system overhead were achieved, looking at error rates did not seem relevant. So, while the total system overhead due to the dataset can be decreased, this is at some additional cost in false negatives. Roughly, however, the increase in error rates should not be much larger than the portion of packets forwarded to the secondary instance, although it is clear from later experiments that additional errors due to session tracking issues will occur with any type of mid-stream packet forwarding.

The key insight learned here is that nothing in respect to packet data or signature costs can be represented by a uniform distribution. Packet sizes can vary widely between different types of activities. Although loose bounds on performance characteristics can be estimated (see Chapt. 4 on page 90), accurate estimates are difficult to pin down. Some detectable activities will have a higher detection cost because they are associated with larger packet sizes. Packet sizes associated with various detected events may also vary over time. The

Chapter 6. Experiments

distribution of packet characteristics will vary over time, sometimes greatly over small time windows, but also due to diurnal effects.

Many of these effects can be easily observed using any amount of captured packet traffic. Additionally, although not part of this first set of experiments, signatures may also have widely varying costs. These particular aspects of detection are leveraged in later experiments, particularly within the Probabilistic Flowbits approach. It becomes clear in later work that potential performance gains are due to non-uniform distributions. Possible future work on optimizations associated with differing packet sizes and signature costs are discussed in more detail in the final chapter on page 144.

6.2.2 Rule-Based

Within this second series of experiments, several different approaches were used, including artificially biasing the dataset and predictor so that we might increase the apparent coverage of a signature-set. These tests used the same approach and experimental methods as the previous set, but added new signatures to the IDS in order to increase packet coverage.

Two different basic configurations were considered: a) splitting HTTP traffic to a single secondary instance; and b) splitting several different streams to multiple secondary instances. The signatures which were crafted were simplistic in nature, generally identifying one of several different types of traffic or activity on the network based on expected keywords. Not all of the custom signatures were useful for rule-based prediction but for completeness are included in App. B.3.

HTTP Traffic Split

In order to experiment with high-coverage signature-sets a set of signatures was constructed to detect HTTP request methods GET, POST, HEAD, HTTP responses, and URIs embedded

Chapter 6. Experiments

within the first 100 characters of the TCP segment. This duplicates work already done by the “http_inspect” preprocessor of Snort, but was efficacious in showing how high-coverage signatures would affect system performance.

Just as in the last set of tests, packets were transmitted from a secondary machine (see App. A.1) using the command-line *tcpreplay* application but at a fixed rate of 10000 packets-per-second². At this packet-rate there was no significant packet loss even for the largest signature-set used. For each test, 1 million packets were transmitted, containing traffic to or from TCP port 80. The HTTP-biased dataset was produced by taking a set of traffic from the corporate dataset and filtering it to contain only port 80 traffic.

The predictor was configured to shunt traffic to an HTTP-only Snort instance (*s1*) when an HTTP-detection signature fired for a connection. Each prediction was configured to timeout within 30 seconds unless new HTTP events for that connection were detected. The experimental setup consisted of two IDS instances: *s0* and *s1* for which combined alert output, error rates, and gain over the control setup is shown.

Test 29a & 29b: Test 29a is the most generic HTTP Split configuration in which all of the HTTP detection signatures are active for both *s0* and *s1* configurations. Test 29b made two changes to the *s1* instance which were tested simultaneously³: 1) removed the 5 HTTP-detection signatures and 2) removed flow-tracking for the remaining signatures.

As Table 6.1 shows, removing the HTTP-detection signatures results in a substantial decrease in relative overhead costs for the *s1* instance. No HTTP-detection alerts are produced for 29b *s1* experimental instance so the predictor does not incur any additional overhead. The small performance gain shown for 29a is not significant. Configuration 29b was essentially equal in terms of control versus experimental overhead costs.

²<http://tcpreplay.synfin.net> - *TCPReplay Homepage* (Source Code - Retrieved Aug 2012)

³The two changes can be measured independently as HTTP-detection events and alert events occur independently under conditions where the system is easily processing all packets

Chapter 6. Experiments

Setup	Instance	Rules	Packets	Events	HTTP/Alert	FP	FN	Gain
29a	Exp. s0	5510	493420	32278	12153 / 59			
29a	Exp. s1	3362	497317	50567	25014 / 263			
29a	Exp. totals	-	990737	82845	37167 / 322	7	87	-19%
29b	Exp. s0	5510	493493	33130	12791 / 66			
29b	Exp. s1	3357	497476	3400	0 / 288			
29b	Exp. totals	-	990969	36530	12791 / 354	6	82	1.3%
29a,b	Control s0	5510	994166	101474	37969 / 599	1	9	
29c	file	5,535	993,901	67,587	42,560 / 909	0	0	

Table 6.1: Test 29a,b: HTTP traffic split [s1 flow tracking, s1 HTTP-detection]

Removing flow-tracking for the remaining signatures also resulted in a small difference in error rates. This is unlikely to be significant, but is somewhat surprising. We should expect that removing flow-tracking from s1 would significantly increase false positives. However, for most signatures the important aspect of flow-tracking is that a connection is known to be established. For any packet forwarded to s1 we can infer that the connection must be established. The initial flow direction (“to_server or to_client”) appears to be relatively unimportant for HTTP-related signatures.

Test 29c: Test 29c attempted to increase the coverage of the HTTP-detection signatures by using a 3-tuple as the connection definition. There were several interesting issues raised in this experiment.

As can be seen from Table 6.2, using a 3-tuple actually decreased the total packets forwarded. We expected to have a larger number of packets being forwarded to s1. This unexpected issue was due to the “wild-card” filtering mechanism of PF_RING being unidirectional. Interestingly, even though fewer packets were forwarded than 29b, we end up with a larger diversity of individual connections being examined by s1. This results in a decrease in false negatives (FN) over the 29a and 29b protocols by over 50%. This suggests that 3-tuple connections provide a more generalized forwarding policy.

Chapter 6. Experiments

Setup	Instance	Rules	Packets	Events	HTTP/Alert	FN	FP	Gain
29c	Exp. <i>s0</i>	5510	678375	26100	7252 / 146			
29c	Exp. <i>s1</i>	3357	239977	461	0 / 442			
29c	Exp. sum	-	918352	26561	7252 / 588	11	38	-32%
29c-bi	Exp. <i>s0</i>	5510	415089	19990	5610 / 67			
29c-bi	Exp. <i>s1</i>	3357	570100	5026	0 / 533			
29c-bi	Exp. sum	-	985189	25016	5610 / 600	12	30	-39%
29c	Control <i>s0</i>	5510	993965	101479	37974 / 599	2	9	
29c	file	5,535	993,901	67,587	42,560 / 909	0	0	

Table 6.2: Test 29c: HTTP traffic split [no *s1* flow tracking, 3-tuple]
 Poor performance is due to high cost of wild-card filtering in the system kernel.

Changing the wild-card filtering options to operate bidirectionally, resulted in a small improvement in the total packets seen by *s1*, but insignificant differences in error rates. The wild-card filtering was also prohibitively expensive. Wild-card filtering increased system overhead due to system calls by as much 60% for each experiment run. Comparatively, the hash-based filtering of other experimental configurations had no measurable increase in system call overhead. This issue is likely fixable within the PF_RING module by using additional hash-tables to perform faster forwarding. These changes to PF_RING are being considered as a possible future work.

Tests 30-32: Tests 30-32 explored the use of various sized secondary signature-sets. Test 30 demonstrates the maximum gain possible for the default signature-set and the set of HTTP-detection signatures used. The *s1* signature-set for this test is empty. All packets which are shunted to *s1* are essentially dropped, resulting in correspondingly high error rates. Each of these tests had HTTP-detection signatures enabled in *s1*, resulting in higher *s1* overhead costs than were strictly necessary.

The error rates shown only count a false negative if all examples of an alert for a particular connection are missed. The absolute error rates for Test 31 [not shown] corresponds closely with the relative sizes of the signature-sets. If the distribution of events were uniform, then the expected false-negative rate would simply be $1 - \frac{|s0|}{|s1|}$.

Setup	Instance	Rules	Packets	Events	Alerts	FP	FN	Gain
30	Exp. <i>s0</i>	5,535	493,364	27,947	10,734 / 53			
30	Exp. <i>s1</i>	0	497,447	0				
30	Exp. sum	-	990,811	27,947	10,734 / 53	0	146	56.7%
31	Exp. <i>s0</i>	5,535	493,569	32,285	12,156 / 60			
31	Exp. <i>s1</i>	452	497,316	49,788	25010 / 13			
31	Exp. sum	-	990,885	82,073	37,239 / 73	1	133	-5.0%
32	Exp. <i>s0</i>	5,535	493,558	32,267	12,153 / 60			
32	Exp. <i>s1</i>	50	497,311	49,798	25,019 / 0			
32	Exp. sum	-	990,869	82,065	37,232 / 60	1	141	-2.5%
30-32	Control <i>s0</i>	5,535	994,123	101,179	38,434 / 598	2	9	
30-32	file	5,535	993,901	67,587	42,560 / 909	0	0	

Table 6.3: Tests 30-32: [small secondary signature-sets]

6.2.3 Traffic Split

From the biased signature-set experiments a few important characteristics were learned. Firstly, high-coverage detection signatures are costly and should be disabled in secondary signature-sets and potentially optimized in primary signature sets. Secondly, performance gains are more costly than expected in terms of error rates. If there is ever a performance penalty in terms of system overhead, then the approach should not be used. The a performance gain is achieved, then the system could potentially process a larger portion of the incoming data. Unused available overhead is wasted when a system is not under high loads and dropping data.

This last insight led to a Test 35h, in which the system was run under high loads. The data-rate was increased to 50Kpps and a single protocol detection signature was used to detect (and then drop) any TCP packet. This is loosely similar to prior work by Papadogiannakis et. al. in which dropped packets are biased to be non-essential to tracking TCP sessions[71]. A fundamental difference in the 35h configuration was that *any* detected TCP session was selectively discarded, both established and non-established. The result is that in many cases all but the first TCP packet is dropped. Interestingly this test resulted in

Chapter 6. Experiments

decreased error rates when the event output was moderated using output filters. Table 6.4 summarizes these findings.

Setup	PPS	Instance	Packets	Event Filter	FP	FN	Loss/Gain
35h-1	50K	Exp.	2,817,300	1 per sec	358	384	46.0%
35h-1	50K	Control	5,374,140	1 per sec	211	355	
35h-30	50K	Exp.	5,611,437	1 per 30 sec	330	174	-3.2%
35h-30	50K	Control	7,384,190	1 per 30 sec	297	228	
35h-60	50K	Exp.	5,863,625	1 per 60 sec	276	251	-1.5%
35h-60	50K	Control	5,955,261	1 per 60 sec	303	327	
35h-3K	50K	Exp.	5,853,907	1 per 6 min	212	299	-3.4%
35h-3K	50K	Control	5,632,039	1 per 6 min	234	355	
35h-3K-slow	10K	Exp.	9,424,949	1 per 6 min	99	511	9.4%
35h-3K-slow	10K	Control	9,992,701	1 per 6 min	92	536	

Table 6.4: Test 35h: Dropping all detected TCP packets for a high-load scenario

Test 35h suggested that the output and predictor overhead costs of the detection signatures were playing a large role in total system costs. The performance clearly depends largely on Snort’s output “Event Filter” configuration. Just as in the HTTP Split tests, the additional detection signatures were added to both the control and experimental configurations. The detection signatures used were not complex. However, the large number of alerts produced could have a significant cost depending on how the output is used.

An additional test was run to determine the extent of the overhead issues. Test 36 examined all of the detection signatures considered for traffic forwarding. Table 6.5 shows the difference in total alerts and traffic detection alerts for a system with no predictive forwarding. The total events produced by the experimental configuration are highly dependent on the high-coverage traffic detection signatures.

Figures 6.3 and 6.4 show the top 10 process times for different output event filtering conditions. The additional output results in significant increases in predictor-related overhead. For the *unlimited* filter setup, approximately 10% of this overhead was due to signature processing costs. The remaining 30-35% was due to overhead of the predictor reading and processing events.

Chapter 6. Experiments

Test	pps	Filter	Setup	Rules	Packets	Detects	Alerts	Loss/Gain
36	10K	1 per 30	Exp.	5,544	9,995,597	21,389	2,617	-7.4%
36	10K	1 per 30	Control	5,505	9,993,677	0	2,658	
36	10K	10 per sec	Exp.	5,544	9,994,046	440,878	2,631	-32.5%
36	10K	10 per sec	Control	5,505	10,000,340	0	2,649	
36	10K	unlimited	Exp.	5,544	9,993,617	937,529	2,604	-46.0%
36	10K	unlimited	Control	5,505	9,996,366	0	2,646	

Table 6.5: Test 36: Measuring cost of I/O for traffic detection signatures

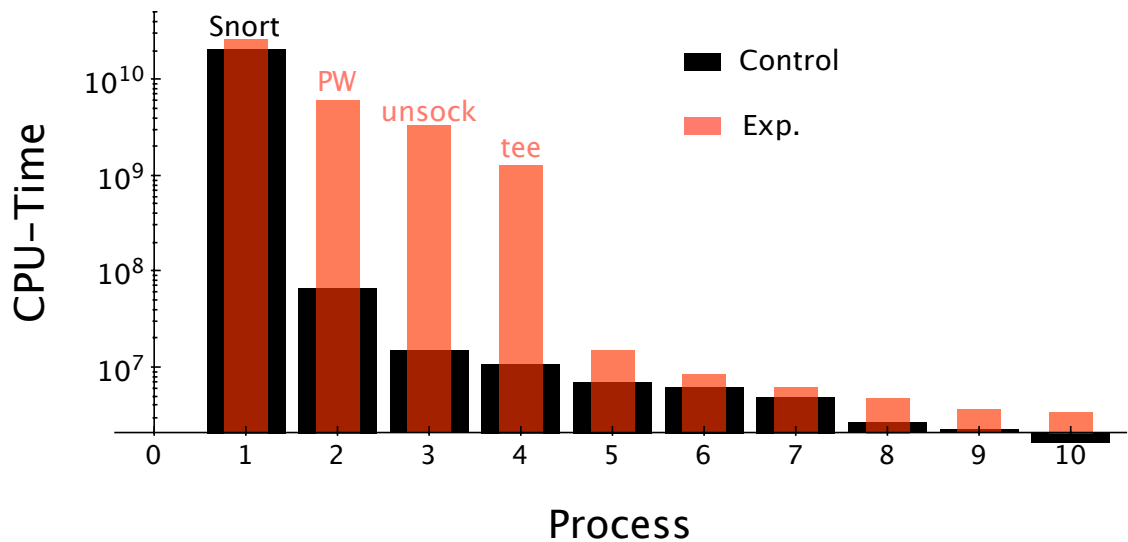


Figure 6.3: Test 36: Process Times showing experimental overhead costs
Event output is unfiltered, thus incurring all I/O costs associated with high detector coverage of input packets.

6.2.4 Discussion

As shown in Chapt. 3, an end-to-end anticipatory approach should be capable of significant performance gains. These gains are only limited by the ability to predict future events and any inherent overhead costs. Unfortunately, such gains are difficult to achieve with existing signature-sets.

For the Snort IDS, coverage of input packet data by standard signature-sets is abysmally low. Increasing packet coverage is the first priority if an anticipatory approach is to work. Artificially increasing coverage by adding new detection signatures can result in moderate

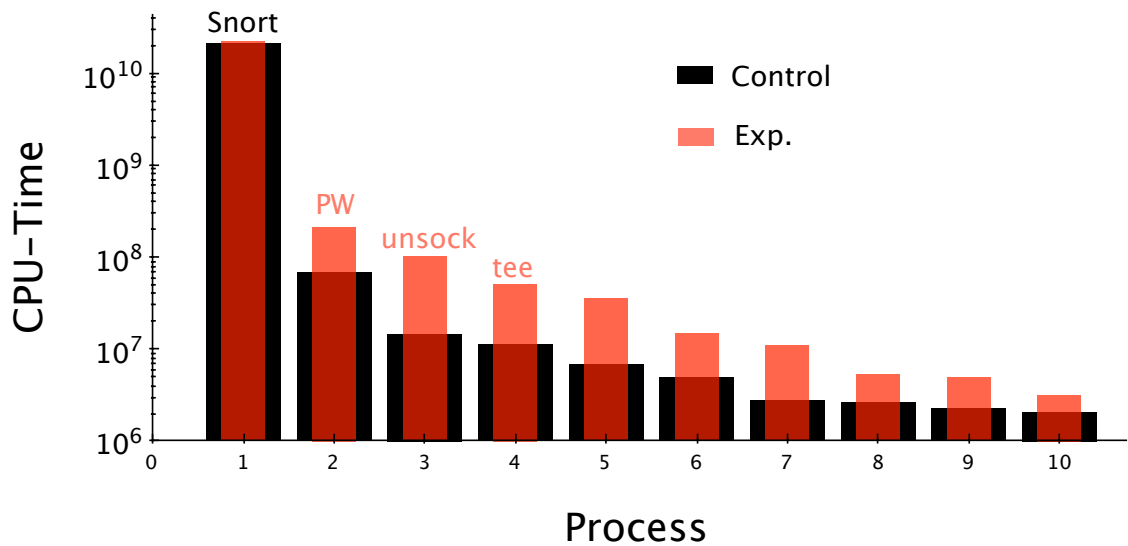


Figure 6.4: Test 36: Process Times with event filter
Event output is filtered at a rate of 1 event per 30 seconds by source IP address.

gains, but at a proportional cost in predictor overhead costs if events are not filtered. In essence, these systems are not designed to have good coverage of input data. Such systems are currently designed to minimize the amount of output. Indeed, the current design of Packet Wrangler falls into the same trap. Its performance is directly dependent on the amount of input.

These characteristics are common within IDS design. Emphasis is placed on assessment of incoming packet data. Since few packets are actually a threat, few events need to be produced. Storing event data for a large portion of incoming packets is not a common design assumption.

The high overhead of the predictor is an outstanding challenge. This might be dealt with using event filtering. Also, many events are not useful for prediction. Careful optimizations could limit to output to events which are actually helpful. Irrelevant events would never be seen by the predictor.

There are a number of other solutions that are worth consideration. Since the cost of producing alerts is relatively high, any event which is not a threat should be dealt with via a

Chapter 6. Experiments

separate I/O interface than the standard alert logging interface. This would allow different subsets of the alert events to be sent to different listening interfaces. In essence we need an IDS which can partition event output to multiple output devices or shared memory regions.

In-memory operations would provide better performance overall, and potentially eliminate most if not all of the predictor overhead costs. As will be shown in the next section, the built-in Flowbits mechanism provides a straightforward mechanism for implementing rule-chaining-based optimizations. These are somewhat similar to the rule-based Packet Wrangler approach, but performed at the granularity of single signatures and implemented using in-memory bit operations.

Sending events via Snort's Unsock interface may also have not been the best design choice. The original datagram is included in the alert events sent over this interface. While this data is never read by the Snort socket server, it might result in later systems with performance issues when this additional data is read for purposes of prediction. Modification of this alert interface to send only alert messages without the associated data might improve performance but would preclude future capabilities based on internal packet features.

6.3 Probabilistic Flowbits

The experiments described in this section were run without any of the Packet Wrangler or related components. The basic setup for transmitting and sniffing network traffic is identical to previous test, as are the measurement methods. The control and experimental configurations generally differ only in the use of Probabilistic Flowbits. For all tests a single Snort instance is used. Each experiment tests either a different signature-set, Snort configuration option, or traffic transmission parameter.

6.3.1 Results

Test 38-1b (0min - 10min)

Test 38-1b tested whether a *Cost-Relevance Policy* with a minimum probability of 0.1 was sufficient for performance gains for a signature set containing an abnormally costly signature. This test considered a packet transmission rate of 10M packets per second. On the test hardware and with the configured signature-set this was sufficient to cause a 20-30% packet loss rate. The control setup had no probabilistic signature activation and all signatures were active. It is important to note that the Cost-Relevance policy probabilities were based on “Qualified Events” for each signature and not the total number of output alerts. This metric was produced by running the Snort’s built-in profiler against the configured signature-set and test data as described in Sect. B.2. Refer to Fig. B.1 on page 166 for a list of the profiler output used for tests 1 and 2.

Figure 6.7 shows the probabilities used for signatures with the “Snort IDs” (SIDS) shown in the left column. The chosen probabilities were based on the first 10 minutes of a network capture. The test data represents the same time window from 0-10 minutes of the same network capture.

Figure 6.5 shows a mean decrease in total CPU-time of 27%. The spike within the first 50 seconds represents startup costs, which are nearly identical between control and experimental setups. As can be seen from Fig. 6.6, decreased overhead results in substantially decreased packet loss (from 25.6% to 0.5%, an improvement of 98%).

Since very few packets are lost we are able to decrease false negatives from 480 to 90 (an 81% improvement) and false positives from 38 to 17 (a 55% improvement). The decreased error rates are all primarily due to lower packet loss. In this case the experimental setup resulted in approximately the lowest packet loss achievable. Clearly, if a default policy were applied (in which the probabilistically activated signatures were removed entirely),

Chapter 6. Experiments

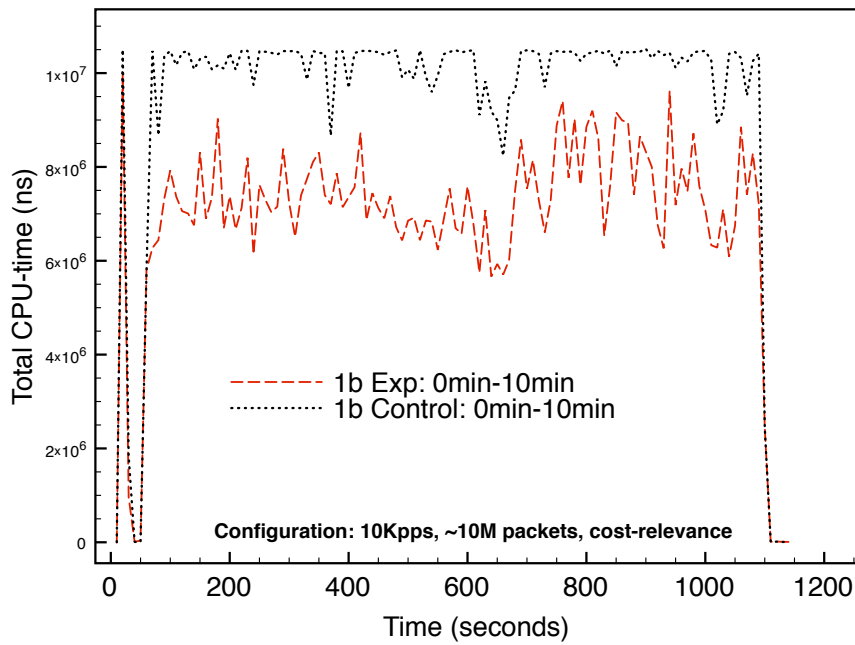


Figure 6.5: Test 38-1b: Total CPU-time for all processes (0min-10min window)

we would expect to incur additional errors due to missing all instances of events associated with the removed signatures.

Test	Setup	Rules	Packets	%Dropped	Events	AlertEvents	FP	FN
1b	Exp.	9831	9947243	0.47%	132941	18107	17	90
1b	Control	9832	7434899	25.61%	77907	13003	38	480
1b	File	9832	9994651	0.00%	135222	18163	0	0

Table 6.6: Test 38-1b: packets processed and error rates (0min-10min)

The discrepancy in the number of signatures is due to a scripting error in signature-set partitioning for the test. Signature 17980 was not present in the experimental configuration for either Test 38-1b or 38-2b, but was not relevant to the results (not a high-cost signature).

Test 38-2b (10min - 20min)

Test 38-2b applied the same parameters as 38-1b to the time window from 10min-20min. It was expected that the adjacent time window would result in poorer performance relative to the first test. The results were somewhat surprising. The total overhead costs in this time

Chapter 6. Experiments

window were significantly better, but suffered substantially worse error rates. Figure 6.6 shows a significant decrease in total CPU-time for the duration of the experiment. The mean decrease in total CPU-time was approximately 36%.

SID	Probability
4677	0.1
9631	0.6
13216	0.7
7980	0.8
11324	0.8
11620	0.8
16067	0.9
16301	0.9
16300	0.9
17166	0.9

Table 6.7: Tests 38-1b & 2b: Cost-relevance policy relative to signature 4677

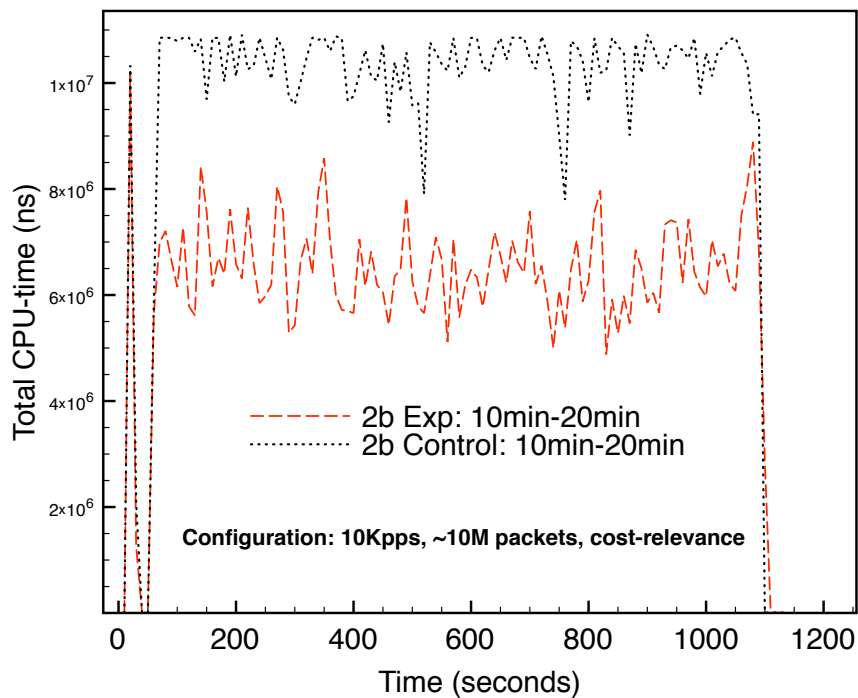


Figure 6.6: Test 38-2b: Total CPU-time for all processes (10min-20min window)

Chapter 6. Experiments

Figure 6.8 shows the packet loss and error rates for Test 38-2b. The performance improvement is still substantial, decreasing false negatives by 63% and false positives by 73%. The experimental setup decreases packet loss over the control setup by about 80%, a smaller gain than test 1b, thus resulting in high error rates. This is a sensible result, significantly decreasing packet loss results in a smaller but still substantial decrease in error rates. However, the large decrease in system overhead likely means that some number of false negatives of the inactive signatures could be regained by using higher signature activation probabilities. Even better error rates might be achieved if the signature activation probabilities were optimal for the characteristics of the packet data in the current time window.

Test	Setup	Rules	Packets	%Dropped	Events	AlertEvents	FP	FN
2b	Exp.	9831	9934762	5.66	127950	15285	13	174
2b	Control	9832	7490043	28.87	74755	10919	49	467
2b	File	9832	10530316	0.00	139426	16667	0	0

Table 6.8: Test 38-2b: Packet processing and error rates (10min-20min window)

It should be clear that if a given signature is of no use to an analyst or when a signature is not relevant for a particular environment, that it should be removed irrespective of its performance. The trade-off being shown is that when a signature is *necessary*, but when performance is degraded due to high signature costs or high traffic rates that good trade-offs can be made in terms of greatly improved performance at the potential cost of a small number of carefully selected increases in error rates.

It is important to note that within both tests 38-1b and 38-2b, some instances of event 4677 were seen for some trials. The high frequency of the signature guarantees that some of these events will be output, even when using a moderately small activation probability. In other words, we decreased the cost of performing detection using the signature but retained the ability to detect it.

Chapter 6. Experiments

Test 38-2j (10min - 20min) - removed signature 4677

This test is intended to show that the probabilistic signature activation provides a middle-ground in terms of performance. Test 38-2j is identical to 38-2b except that signature 4677 is removed from the control and experimental signature-sets and the activation probabilities are re-scaled based on the Avg/Check cost for signature 9631. This has the obvious effect of greatly reducing the total overhead on the system. As a result, at the configured packet rate of 10Kpps, no packet loss is incurred and no performance gains are achieved by the experimental setup. Additional tests at high packet rates were performed and no performance gain was seen for any tests using the Cost-Relevance activation probabilities.

SID	Probability
4677	0.0
9631	0.1
13216	0.3
7980	0.5
11324	0.6
11620	0.7
16067	0.7
16301	0.8
16300	0.8
17166	0.9

Table 6.9: Test 38-2j: Cost-relevance policy with re-scaled probabilities
Probabilities were re-scaled after removal of signature 4677.

Test	Setup	Rules	Packets	%Dropped	Events	AlertEvents	FP	FN
2j	Exp.	9831	10487270	0.3917%	135341	16303	12	111
2j	Control	9831	10485319	0.4103%	135348	16142	13	116
2j	File	9831	10528515	0%	139329	16661	0	0

Table 6.10: Test 38-2j: Packet processing and error rates after signature removal
Showing results within the 10min-20min time window after removal of signature 4677.

An astute reader may suspect that removing the obnoxiously costly signature would result in a set of activation probabilities which would have little effect on overall performance. Figure 6.11 shows potential gains for each of three policies considered. For the

Chapter 6. Experiments

Cost-Relevance policy the gain for the training set is always less than 4%, which, for our purposes results in no detectable performance gain for the test data.

In theory, the Cost-Relevance policy should provide good generalizability. However, it does not adequately leverage the actual expected costs of each signature within a small time windows. From the figure it is clear that performance gains are still achievable using the other activation policies. For the signature-set being examined and a minimum probability of 0.1, the gains for the training set for the Relevance and Cost policies would be approximately 10% and 30% respectively. Interestingly, due to the Cost-Relevance

		Minimum Prob	0.5	0.4	0.3	0.2	0.1
Cost-Relevance (Avg/Check)	Relative decrease		2.19%	2.77%	3.15%	3.40%	3.79%
	Absolute decrease		1738714	2199116	2499255	2700492	3005179
Relevance (Checks)	Relative decrease		5.14%	5.78%	7.00%	8.13%	9.42%
	Absolute decrease		4075633	4585497	5551775	6452848	7472265
Cost (Microseconds)	Relative decrease		16.43%	19.86%	23.57%	26.54%	30.51%
	Absolute decrease		13038600	15759808	18698795	21054724	24207709

Table 6.11: Calculated gains for various signature activation policies
Calculations are based on Snort profiler output when using the 0min-10min training set.

Policy, other than signature 4677, the set of probabilistically activated signatures within Tests 38-1b and 38-2b actually had nothing to do with the achieved performance gains. For the training set, the remaining signatures accounted for only 0.6% of the total CPU-time incurred in processing the dataset. However, by using a different policy, we can still achieve good performance gains. In particular, the Cost policy seemed the most obvious candidate.

Test 38-2k (10min - 20min) - 40Kpps - Cost Policy

Test 38-2k modified the setup for Test 38-2j by applying the Cost policy and increasing the packet transmission rate sufficient to cause significant packet loss. The probability table for the Cost policy signature activations of Test 38-2k are shown in Fig. 6.12

This experiment was intended to show that even in the case where no obvious candidate for signature removal can be made, performance gains can be achieved by employing

Chapter 6. Experiments

probabilistic flowbits. However, it appears that when a system is completely saturated that probabilistically activating signatures has little effect even though there should be a decreased cost. The incoming packet rate is large enough that relatively few packets are being processed. This is consistent with early tests performed using high-cost signatures such as 4677. When packet losses were over 50% gains were diminished or eliminated with only a minor difference in error rates. Error rates for Test 38-2k are shown in Fig. 6.13

SID	Probability
8701	0.1
17512	0.3
14990	0.4
2707	0.5
16014	0.6
17513	0.6
13216	0.7
12183	0.7
17166	0.8
630	0.8
623	0.8
11324	0.8
624	0.8
3550	0.8
621	0.8
622	0.8
2580	0.9
14989	0.9
5712	0.9
12591	0.9
4136	0.9
2442	0.9
7047	0.9
16521	0.9
2570	0.9
619	0.9
11272	0.9
2705	0.9
11273	0.9
4135	0.9
16743	0.9
11196	0.9

Table 6.12: Test 38-2k: Cost policy probability table

Test	Setup	Rules	Packets	%Dropped	Events	AlertEvents	FP	FN
2k	Exp.	9831	6656867	36.784%	57299	10663	179	578
2k	Control	9831	6371688	39.492%	50918	10030	180	619
2k	File	9831	10530292	0.000%	139366	16663	0	0

Table 6.13: Test 38-2k: Packet processing and error rates using a Cost policy
Showing results for the 10min-20min time window.

6.3.2 Analysis & Criticisms

Benefits notwithstanding, there are a number of potential criticisms of the signature activation approach. We might argue that irrespective of whether the highest-cost signatures are removed, that there is usually a set of activation probabilities and a data rate at which performance gains can be seen using our approach. However, if the policy metric were uniformly distributed, it would be difficult to make acceptable trade-offs between signatures. Additional information would be needed (such as the use of a weighting function to prioritize signatures based on importance). Although it is unlikely for signature cost or frequency to be uniformly distributed, this is exactly the situation desired by IDS designers. It is also the situation which occurs after some number of signatures are probabilistically “flattened”. In brief, we can only really apply this technique for the high-frequency and high-cost signatures. As soon as we have lopped off the head of the curve, we are left with the long tail of the distribution and no more gains will be possible.

Another criticism is that we have yet to show how the approach can improve performance when there is no packet loss. The initial arguments made suggest that for frequent events we might gain performance by checking infrequently. The issue with this argument is that events which occur frequently are often very cheap to detect. If the approach is only relevant for systems which are losing data, then the benefits would only be seen during infrequent high-load events. As a counter to this criticism, we can use probabilistic signature activation to add new signatures that otherwise would have been prohibitively expensive.

Chapter 6. Experiments

Our long-term goal is to improve performance irrespective of system load. This is more difficult to achieve than one would expect from a cursory analysis. Retrospectively, this is not surprising as the performance of an IDS is dependent upon the signatures used, tuning of myriad parameters, and highly dependent upon the incoming data. Since our model ignores most of this complexity, we can expect significant gaps in our ability to predict performance characteristics of an activation policy.

The results thus far suggest that exploring new ways of applying the current set of activation policies will be worthwhile. The approach is likely to be important for a number of scenarios which have not yet been addressed experimentally, such as cases where flooding or other denial of service attacks actively disable IDS systems to mask more nefarious activity, or where the cost of detection precludes complete coverage of known attacks. The demonstrated approaches might easily overcome such limitations without significant system modifications. For example, a flooding attack would cause cost and relevance of a set of signatures to change, resulting in decreased sampling rates for the offending signatures and minimizing the impact of the attack. Any relevant events would still be seen but would maintain a more or less constant overhead cost.

A final criticism deals with dataset characteristics. It seems intuitive that signature costs and frequencies be non-uniformly distributed for large networks and large datasets. In all signature-sets examined, the signature costs and frequencies appear to be power-law distributed. However, whittle away long enough at a signature-set and we will be left with a more or less uniform distribution which will be difficult to optimize in the ways described. In the end, the signature-based optimizations are limited by distributional characteristics and often only apply in high load scenarios. Nonetheless, that these optimizations may be precisely what are needed to achieve the broader goals of increasing coverage and decreasing costs.

In counterpoint to the above criticisms, there are also other potential benefits of the approach. The Probabilistic Flowbits approach presents a possible solution to Flowbit

Chapter 6. Experiments

evasion issues identified in 2012. In their work, Tran et al. demonstrated shown that the current “Flowbits” mechanism is vulnerable to evasion attacks[88]. Within most IDS the state of TCP and UDP streams are tracked for the duration of each connection. Flow tracking enables signatures to be made active based on the state of the stream and prior signature activations. Tran et al. provide both an approach to automatically generate evasion attacks and to generate signature-set patches which guard against the generated evasion attacks.

While the current Probabilistic Flowbits approach probabilistically disables signatures, evasion attacks could be thwarted by probabilistically activating signatures even though their flow state preconditions have not been met. The evasion attack becomes far more difficult as the attacker now must deal with their evasion failing with some potentially unknown probability.

There are also an array of more complex examples related to threat detection within application-level protocols. In many cases, accurate tracking of application state is prohibitively costly and protocol state cannot be known explicitly. Extending the Probabilistic Flowbits approach, *Probabilistic flow tracking* would enable application-level signatures to be activated based on the probability of a particular application state. Exploration and experimentation of this and other possible uses of the approach is left to future research.

Chapter 7

Impact, Conclusions, & Future Work

The Packet Wrangler approach has been far less fruitful in terms of producing a useful technology than we had initially hoped. The basic model is elegantly simple-minded. Online prediction of attack sequences is not that difficult. But online refinement of an active IDS is fraught with engineering obstacles. Existing systems do not expect their data-streams to be manhandled. Indeed, the simple-minded approach suffers some of the same challenges as the IDS which we are attempting to optimize.

Gladly, there appear to be many ways of using learned statistics to improve performance. The Probabilistic Signature Activation approach presents only a single type of optimization. However, the maximum performance gains using these approaches are likely to be bounded by the cost of the signatures which are being optimized. Practical maximum gains can be explored by simply removing highest-cost signatures and examining behavior under different network loads. As such, the Probabilistic Flowbits approach represents a middle-ground between arbitrary packet drops and complete removal of high-cost signatures.

The theoretical performance gains of the Packet Wrangler approach are not bounded in the same way, but depend directly on the characteristics of the signature-sets used and the data being assessed. The practical limiting factors are predictor performance and overhead

Chapter 7. Impact, Conclusions, & Future Work

costs. Some of these costs might be eliminated by modifying the IDS architecture so that superfluous operations are not performed. Although the Packet Wrangler prototype has shown promise, the total coverage of existing signature-sets is quite small. In the experiments conducted the overhead costs generally swamp any performance gains.

We are actively pursuing a number of additional ideas to address the flaws in the Packet Wrangler approach. In particular, Probabilistic Flowbits partially addresses the high overhead costs of high-coverage signature-sets. Packet Wrangler currently computes signatures and fires alerts for a large number of incoming packets (the definition of high coverage). In respect to our thesis, this is exactly opposite to the problem in which too little information is retained by the IDS. A large number of superfluous events which do not gainfully assist the predictor still consume system resources. We expect, but have not yet shown, that combining these two approaches will improve performance by optimally increasing coverage.

Our attempts at end-to-end anticipatory performance optimization have also led to a now obvious fact. We cannot use anticipatory optimization directly coupled with unmodified detection system. The design assumptions conflict with our goals. Not enough information is gained for each data input. Our systems do not know how to perceive the ebb and flow of network traffic beyond simple state-machine emulation. Our systems only perceive pin-pricks of threat in the dark, each one violating our assumptions of bubble-boy sterility. Before we can achieve gainful anticipatory optimizations, we need to maximize the amount of information gain and minimize the amount of work needed to do so.

As discussed in the Related Work section on page 40 there are systems and models which maximize the information gained for each available input. Such approaches attempt to expose the meaning of network attacks in the context of network topology, machine characteristics, and learned prior knowledge. The basic tenant is to provide relevant context for understanding network threats. While powerful, simply conveying context is not enough. Such systems are incredibly expensive to maintain. Their costs generally greatly outstrip

Chapter 7. Impact, Conclusions, & Future Work

their benefits. Just as is the case with modern IDS, these types of computer defense systems do not anticipate future knowledge. There is no anticipation, just a direct application of deductive reasoning. It would seem to me that anticipatory optimizations is almost stupidly relevant.

In summary, this dissertation describes and demonstrates both a rudimentary anticipation model and a probabilistic middle-ground between removal of high-cost signatures and partial inclusion via probabilistic activation. The signature activation approach provides a means of dealing with moderate packet loss by holding high-cost signatures accountable. Both approaches have a straightforward theoretical basis which suggests that future system will all eventually include such optimizations. Clearly, the probabilistic signature activation approach is straightforward enough that it should be a fundamental aspect of any modern signature-based IDS.

7.1 A “Household Survey” Analogy

If we were giving surveys by walking door-to-door we might be confronted by some of the same problems as occur in signature-based intrusion detection. Assume that we have a single exhaustive survey of yes/no questions (our signature-set). For our survey, when a household answers “no” to all questions, then we make a check-mark and move to the next home. However, when a household answers “yes”, then we take the time to write down their demographic information (an IDS alert).

Our plan is to visit every house in sequence from our starting location (a packet sequence). Our goal is to visit every house and to ask every survey question, but we have a limited amount of time. When we have a short survey or few houses to visit, this is easy. But when we have a long survey or many houses, we have a problem. We will not be able to finish before we will have to give up and start skipping houses. We might miss

Chapter 7. Impact, Conclusions, & Future Work

entire neighborhoods and demographics. Our survey results will not be representative of the population in these neighborhoods.

If we know that we will not have enough time, we can choose subsets of signatures to skip based on their cost. If we have a large number of houses, then we will ask high-cost questions infrequently. We will sometimes miss out on opportunities to learn about the attitudes of some households on complex questions, but this is an acceptable trade-off. This is the basic idea behind Probabilistic Signature Activation.

What would be even better, however, is if we had some prior knowledge over which questions might be relevant for a subset of households. As an example, perhaps there are two different clubs to which people on our survey route subscribe: the Environment Club and the Gun Club. The first few surveys given in our neighborhood give mostly tree-loving-hippie answers. It would probably be OK to eliminate Gun Club questions and focus on Environment Club questions for next few houses. We might miss a few opportunities, but we have saved a lot of time. This is essentially the idea behind the Packet Wrangler approach.

In the Probabilistic Signature Activation optimization, it is straightforward to figure out which questions to skip based on cost. The longer a question is, the longer it takes me to ask. This optimization works regardless of whether the house answers a question. If we ask the question at all it costs me time. If we skip some questions, we gain some time.

It is also straightforward to predict simple sequences for the Packet Wrangler optimization. In most cases, if we get a positive response for one household, the next one will likely give a positive response. However, within our analogy, most households (packets) do not give any “Yes” responses for any question. For most households, we ask all of our questions, but never record any demographic information.

In our analogy, the survey (signature-set) is composed of very specific questions. We are asking questions that are analogous to whether a household participates in some nefarious

Chapter 7. Impact, Conclusions, & Future Work

activity (like dumping toxic waste in the sewer). Even if all households always answered truthfully, most households would truthfully answer “no” to every question. We are never asking a general question like “Are you an environmental advocate?”, of which perhaps a significant portion of households might answer “yes”. This is not the specific information we are after so we do not ask it. But this is exactly the sort of information we would need to anticipate which questions the next household might answer affirmatively.

This is the basic problem with anticipatory optimization approaches with the signature-sets and design assumptions of many detection systems. For most inputs we do not gain any information. When we force the issue by adding general questions (i.e. expanding packet coverage), the information we gain costs us too much.

This leads us to an insight regarding current detection approaches. We can get performance gains using anticipation, but first we need to make anticipation cheap. These systems are designed to learn the minimum amount of information necessary for detection for the minimum cost. What we need are systems which maximize the amount of information gained while minimizing the cost per bit learned. We can still constrain the information gained to be those things that are useful, but we need more than just an identification of threat. We need to be able to cheaply, even optimally identify features which are useful for prediction.

7.2 Biological Metaphors & Future Work

First and foremost, the coverage of future systems must be greatly expanded. Existing systems provide a myopic view of threats with most of the context and most of the detection still living in the minds of network threat analysts. Many of us have worked closely with systems which gather general information concerning network traffic and networked machines. This gathered knowledge is exactly the type of expanded coverage that is needed

Chapter 7. Impact, Conclusions, & Future Work

to perform prediction and anticipatory optimization. But such systems are not yet cheap. We will need to bootstrap their performance and anticipatory optimization presents a possible solution.

In addition to those explored in this dissertation, there are a number of other biologically inspired optimization approaches that are tantalizingly accessible. A couple of concepts in particular are simple enough to be easily implemented as extensions to the Snort IDS. One concept applies intracellular dynamics ideas to the task of limiting individual signature activation rates. The insight is simply that once an event occurs (or even once a signature is checked) the signature should be held in a “recovery” or “refractory” period for some amount of time. This is similar to the way that neurons will fire and then slowly recover as ions are pumped across the cell membrane. The refractory period prevents the neuron from firing too frequently. For detection systems, this would provide a useful optimization across all signatures. The challenge is in having good distributional statistics to ensure that the refractory period for each signature is optimal.

Another biologically-inspired approach uses inter-cellular communication and network concepts to either inhibit or stimulate “adjacent” signatures. This is somewhat similar to the Packet Wrangler prototype, but much simpler and focusing on small deviations from the default signature activation policy. Adjacency in such an approach might be decided in a number of different (and possibly overlapping) ways. Taxonomic relationships between signatures provides one method, although the sparsity of current taxonomies may result in too few degrees of separation between signatures. Other adjacency approaches might directly make use of a trained Bayes net or neural network which reflects correlations between events. Being able to slightly bias the way that a signature is used was sorely missing from the Packet Wrangler approach, which requires a high-confidence decision on future states. Allowing for small deviations from default behavior allows a wider array of behaviors and simple optimization strategies.

Chapter 7. Impact, Conclusions, & Future Work

Simpler forms of anticipation such as priming are also relevant. If we had recently activated a signature we might cache some portion of the packet and store it based on a cheaply computed hash or checksum. For future packets we would have a hint that there was a potential match. Alternatively, future packets which match the hash would not necessarily need to be assessed at all. We might dumbly fire the signature even though the hash may be incorrect. The probability of a possible false positive would be directly related to the computing cost and uniqueness of the hash. But in some cases, incurring a false positive may provide superior results than being safe and always computing the signature.

There are also possibilities which more closely related to the “bubble-boy” analogy discussed in the introduction. In a conventional IDS approach, some efficiency is achieved by gaining “just enough” information to identify a minimal set of events. Existing systems have a checklist of items which are guaranteed to be examined. Outside of this short list, the IDS knows nothing. This differs fundamentally from biological immune systems in which an incredibly large number of patterns (both malicious and benign) are detected precisely[40]. Many interesting parallels between biological and computer immunology have been explored in great detail within the last two decades. Although these topics have not been a focus within this dissertation, a couple of concepts are particularly germane.

One characteristic which seems particularly relevant is that of a continually changing and adapting immune response. It is not just that new antibodies are continually being created, but that a vast reservoir of potential immune responses are always readily available depending upon what is currently being detected[40]. In our signature-based detection engine, it would be as if we had millions of patterns which are each active in proportion to the currently active (or potential) threats. Indeed, a more refined implementation of the Probabilistic Signature Activation approach could be used to emulate this behavior. Rather than resources used for detection depending primarily on the signature, they are dependent almost solely on the active or predicted threat. Indeed, as detection algorithms

Chapter 7. Impact, Conclusions, & Future Work

become more refined, it is likely that all signatures have essentially the same per-packet cost. Biasing the detector in this way will become necessary.

Another characteristic which has been mentioned previously is that of safety, or as Stephanie Forrest et al. described as “imperfect detection” in their 1997 article on Computer Immunology[40]. It seems impossibly unlikely that biological immune systems perform any type of “perfect” or “safe” pattern matching. Biological systems also make mistakes, resulting in allergy and autoimmune disease. Yet with relatively few resources most threats are identified and thwarted in time sufficient for an organism’s long-term safety. This is paralleled within modern IDS in the sometimes high costs of false positives. These costs can be internal to the IDS, such as a poor signature demanding resources which would be better spent detecting more important threats, or external, such as a flood of duplicate alerts which clutter the displays of a computer analyst.

In light of our exploration of various anticipatory optimizations, imperfect detection is more than just an obstacle to overcome. It is also a tool which can be used to achieve efficient detection. We might even suspect that biological systems have evolved to work in imperfect ways exactly for the purposes of efficiency. The strategies which we use to “tune” our detection engines need to consider *how and when to allow the right kinds of mistakes*. Every possible way to give up safety may be an opportunity for more efficient detection. If we choose to never make mistakes, we miss opportunities, and we will sometimes lack the finesse necessary to succeed.

Criticisms

Reading the literature sometimes seems to suggest that we must always hold true to a particular analogy when comparing biological systems and computing systems, but this is not necessary. There are far too many fundamental differences for all of the details to be relevant. Our desire for both perfection of design and the purity of our metaphors

Chapter 7. Impact, Conclusions, & Future Work

often make for good papers, but not necessarily good implementations. Instead, we are left to learning about basic biological and cognitive mechanisms and applying and mixing these mechanisms within our designs and implementations. While many of the approaches discussed in this dissertation were presented in the context of a biological metaphor, there was never a need to hold true to the metaphor in the implementation. It is hoped that the reader accepts our need to be inspired rather than a need to model that which inspires.

The implementations presented are also fairly rudimentary prototypes. More work is needed to expand the prototypes into useful augmentations of detection engines. One aspect in particular which is missing from the prototypes is any type of online learning or adaptation. While our tests did not assess learning approaches, they are clearly applicable. Future prototypes will incorporate both online learning and adaptation as well as more sophisticated statistical methods. As mentioned previously, there are a number of optimizations which could be achieved if better distribution statistics were known. As usually begets the experimentalist, we are limited by the datasets available to us. There are still many uncharacterized trade-offs for future studies (such as the trade between detection cost and increases in detection latency).

A final criticism deals with dataset characteristics. It seems intuitive that signature costs and frequencies will often be non-uniformly distributed. In all signature-sets examined, the signature costs and frequencies appear to follow power-law distributions. However, whittle away long enough at a signature-set and we may be left with only the long tail of the distribution, which will be difficult to optimize in the ways described. In the end, the signature-based optimizations are limited by distributional characteristics and often only apply in high load scenarios. Nonetheless, these optimizations are precisely what is necessary to achieve the broader goals of increasing coverage and decreasing costs.

Closing Thoughts

It should be obvious that existing detection approaches fall flat. We cannot always detect what we need to detect when we need to detect it. The approaches we use are too safe, but this safety costs us too many computing cycles. In a way, we should give up. We should give up the guarantee of safety. We should detect what we can detect when it is convenient. The leaves rustling in the forest do not mean anything to our detection systems because they are not designed to consider the future. Our systems respond to threats. Our systems sometimes react to threats. But our systems must instead predict threats, respond proactively, and to choose what to detect on a hunch.

Modern detection systems are paralyzed by the mere possibility of predators, always searching for the threat and not the precursor. We attempt to make them resilient to predators by hardening software, by digitally signing our transmissions, and by dumbly performing brute-force searches for viruses and malicious code. But quite often we are already in the water with piranha. We did not notice the blood in the water before we dove in. We were not looking for it. Because we do not proactively respond to potential threats, we will not know about our fishy friends until they have already taken their first bite.

Many of us dream of a future in which our computing systems are as robust and adaptable as biological organisms. We would like for our computing machines to be as resilient as biological systems. Yet we often insist on protecting our systems with simple-minded border security. During a human lifespan we are infected with billions of organisms representing millions of different variations. If our skin were our only barrier to infection, we would perish at the first infection, and our life would end in infancy.

It seems that our computing machines are at an impasse. Current systems have a kind of bi-modal safety, all or nothing. We must abandon the fantasy of perfect security and allow our systems to make acceptable errors. The expense of maintaining the facade of bubble-boy security is eventual and certain failure.

Chapter 7. Impact, Conclusions, & Future Work

It would be nice to be able to claim that this research will revolutionize the way that IDS is performed, indeed, how computer network defense is performed. However, this work has been more exploratory than revolutionary. The problems examined and the solutions championed are nothing new to the initiated. Most of the techniques for performing anticipation are old hat to any artificial intelligence buff. However, we must change our thinking about what it means to be secure. A secure system is not one that is unplugged from the world. A secure system is one that is resilient in the face of countless threats while being steeped in the milieu of rich computational and networking environments.

Just as a willow will bend in the force of a summer storm, our computing systems must be capable of a pliable resilience in the face of this threat. Until we find the right combination of techniques, technologies, and algorithms, we will continue to suffer the consequences of our digital fragility.

Appendices

A	Hardware & Software Configurations	156
B	Snort Configurations & Analysis	164
C	Issues & Confounding Factors	172
D	Scripts & Algorithms	176

Appendix A

Hardware & Software Configurations

A.1 Test System Configuration

The principle test system was a Linux Ubuntu 11.04 system with kernel 2.6.35-30 SMP, running with 28GB of main memory on a pair of 2GHz Dual-core AMD Opteron processors. The system was configured with two consumer-grade Broadcom gigabit ethernet cards. One network card was used for system management while the other was used for sniffing and forwarding traffic. Table A.1 summarizes the system hardware and operating system configuration. A secondary system was used to transmit traffic using the “tcpreplay” command. This system was almost exclusively an Apple iMac Intel i7 running OSX 10.7 was configured with a standard gigabit ethernet card and 16GB of main memory.

Snort required some additional options to enable access to the PF_RING data-acquisition (DAQ) module:

Listing A.3: Snort configuration options used for compilation

```
./configure LIBS="-lpthread -lpfring -lpcap"  
  --with-libpfring-includes=/usr/src/pf_ring-4  
  --with-libpfring-libraries=/usr/local/lib  
  --with-daq-includes=/usr/local/include
```

Appendix A. Hardware & Software Configurations

System	Component	Description
Server	System mainboard	Tyan S3992
Server	CPU	2x Dual-Core AMD Opteron 2212
Server	Main memory	28GB
Server	Operating System	Ubuntu 11.04
Server	Linux Kernel	2.6.35-30-generic #61-Ubuntu SMP x86_64
Server	Network Interface	2x Broadcom NetXtreme BCM5780 Gigabit Ethernet
Network	iMac	Broadcom BCM57765

Table A.1: System Hardware Configuration

Software	Version
Snort	2.9.1.2 IPv6 GRE (Build 84)
libpcap	1.1.1
libdnet	1.12
tcpdump	3.4.4 (build 2450)
PCRE	8.12 2011-01-15
ZLIB	1.2.3.4
PF_RING	SVN version 5074

Table A.2: System Software Configuration

```
--with-daq-libraries=/usr/local/lib/daq
--enable-ipv6
--enable-zlib
--enable-normalizer
--enable-perfprofiling
```

Additionally, when running Snort using the PF_RING Data Acquisition (DAQ) module, pointing the running instance to the location of the DAQ module was necessary:

Listing A.4: Snort command using the PF_RING data acquisition module

```
snort -i eth0 -c /usr/local/etc/snort/etc/snort.conf
--daq-dir /usr/local/lib/daq
--daq pfring
```

However, for many tests the PF_RING /glsdaq was not used due to automated filtering mechanisms cleverly implemented to allow whitelist and blacklist IP addresses based on

Appendix A. Hardware & Software Configurations

Snort alerts. The basic command used in most experiments is shown in Listing A.5. See the Snort man pages for information about the options used. Basically this configuration uses a Unix socket for output, uses Coordinated Universal Time (UTC) time stamps, disables binary output, and sets an instance number for alert IDs.

Listing A.5: Snort command for experiments

```
snort -i eth0 -c <snort.conf> -N -U -A unsock -G <instance>
```

On the network machine the *tcpreplay* command was run with options to modify the packets transmitted per second. The total number of packets transmitted was determined by the total size of the packet capture file being read. For most experiments delaying packet transmissions for 60 seconds prior to transmitting packets was sufficient for startup of the IDS. A similar amount of time was provided to allow the system to settle after the experimental run. During these “rest” period, system performance was monitored. This provided an indication of baseline, startup, and stopping costs.

Listing A.6: TCPReplay command

```
tcpreplay -i <interface> -l <packet_rate> <file>
```

A.2 Training and Test Data

In general, network traffic training data is extremely difficult to obtain. The datasets which are readily available have a number of significant issues which generally preclude their use in experiments. The 1998 dataset, for example, consists of generated traffic which represents an arbitrary selection of network scans and attacks. The distribution of attack traffic to normal traffic is not realistic, which is necessary for showing that under realistic conditions, anticipatory bias can be gainfully applied. Other researchers have found similar issues with the DARPA dataset[20, 21, 45]. This dataset was used for testing and limited

Appendix A. Hardware & Software Configurations

experimental purposes, and in particular for determining if dataset had an affect on scaling performance. Results are consistent with results using a corporate dataset, but have generally been excluded from analysis.

Experiments to date have used several sets of corporate network traffic, which represents hundreds of gigabytes, yet still presents a number of outstanding challenges. The dataset was collected on the outside of our corporate firewall, representing all connections made to external computers and networks. Internal traffic is not part of the data collected and as such, many types of insider attacks are not present at all. The benefit of this collection is that it does not limit attacks to those that are able to pass through our firewall. Even when an attack is stopped at the firewall, the packet traces remain. An unsuccessful attack on one port will often lead to a series of attacks on adjacent ports or portions of our corporate domain. Many of these useful correlations would not be seen if data were collected inside our corporate firewall.

Another issue with the current corporate dataset is that all web traffic is seen through a web-proxy that resides inside the firewall. As such, not every request made even passes through the proxy, being served by the proxy itself. Those connections that do require sending requests to external servers are all requested from the single internal IP of our web-proxy. This is not a problem when the full 5-tuple of a TCP connection is used to define a *connection*. However, it is useful to expand the definition of connection using the 3-tuple of protocol, source IP, and destination IP. Most TCP connections are short-lived, resulting in short-lived predictions. Using a broader definition for connections allows correlations to be seen outside of a single TCP connection. Statistics can be collected which allow temporally adjacent TCP connections between the same pair of IP addresses to be used for prediction.

The corporate dataset was partitioned into chunks representing approximately 10 minutes each (several gigabytes in size). This was done in order to run training and testing on datasets which were adjacent in time, while maintaining the ability to easily run an experiment for many trials with the same or differing parameters. For the experiments in

Appendix A. Hardware & Software Configurations

this paper, the first 10 minutes of traffic were used for training and segments from 10-20 and 20-30 minutes were used for testing. The transmitted packet rate was modified via a parameter passed to the TCPReplay command line program. Issues with any corrupted data were ignored as they would occur in both the experimental and control setups.

A.2.1 DARPA 1998 Training Data

Because the dataset was available, some initial tests used the dataset for software testing. In particular, prior to proposing this research some time was spent generating Naive Bayes predictors using the first week of DARPA training data from the original 1999 DARPA Intrusion Detection Evaluation[62, 63]. These initial tests are included here for completeness and in order reference within other chapters of this dissertation. There are a number of issues with this dataset which have been widely publicized within the last decade. Although illustrative in limited cases, the dataset presents an unrealistic distribution of network traffic, individual connections, traffic characteristics (such as Time to live (TTL) being characteristic for attacker traffic), and other traffic patterns[20, 22, 38, 67].

sid → sid	Occurrences	Description
469 → 469	13997	ICMP PING NMAP → ICMP PING NMAP
1620 → 1620	478128	Non-Standard IP protocol → Non-Standard IP protocol
1620 → 13949	123893	Non-Standard IP protocol → Spoof of domain
1620 → 15934	48766	Non-Standard IP protocol → DNS for 172.16/12
1620 → 15935	13821	Non-Standard IP protocol → DNS for 192.168/16
13310 → 2925	17638	Apache DOS attempt → Web bug 1x1 gif attempt
13310 → 13310	138106	Apache DOS attempt → Apache DOS attempt
13948 → 1620	24901	DNS cache poisoning → Non-Standard IP protocol
13949 → 1620	109765	Spoof of domain → Non-Standard IP protocol
13949 → 13948	24888	Spoof of domain → DNS cache poisoning
15934 → 1620	54953	DNS for 172.16/12 → Non-Standard IP protocol
15935 → 1620	14923	DNS for 192.168/16 → Non-Standard IP protocol
	1,049,782	

Table A.3: High Occurrence Alert Sequences for the 1998 Dataset

Figure A.1 shows a stochastic matrix generated from 360,591 alerts from Snort's specialized preprocessors prior to clustering. The first column of this matrix represents events for which no subsequent event was seen. The first row represents events for which no

Appendix A. Hardware & Software Configurations

prior event was seen. The diagonal represents events which predict sequences of identical events (e.g. scans, malformed packets, ICMP activity, statistical threshold violations, etc.). The built-in event generators for Snort represent many of these classes of events. For the purposes of this research, we can ignore these stateful detectors (and stream processors) and focusing on the non-stateful detection which represents the bulk of the signature-set and computing cost.

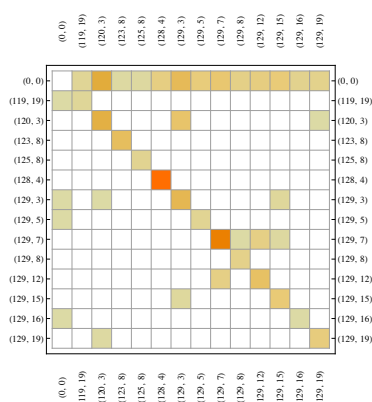


Figure A.1: Stochastic matrix set showing Snort's built-in preprocessors

This matrix was generated using the DARPA 1998 training dataset and Snort's default alert generators (genid,sid) and the N ext temporal semantics. The y-axis are event priors and the x-axis the consequents. The first row and column are events with no priors and events with no consequent respectively.

Figure A.2 represents a stochastic matrix produced over the entire DARPA 1998 training dataset. This dataset represents approximate 3GB of capture packet data (13,620,149 packets) which results in 1,096,099 packet-level Snort alerts using a recent release of the Snort VRT signature-set (ignoring stream and specialized preprocessor alerts).¹ Of particular interest are the events which show a high correlation with future alerts for the same connection tuple. Table A.3 describes all sequences which occur more than 10,000 times.

These events account for 95% of the alerts. At least for the dataset, we have a small set of superb predictors which account for almost the entire set of alerts and predict temporal

¹<http://www.snort.org/snort-rules> - Snort Homepage (SourceFire® "VRT" signature-set - Retrieved on Oct. 17, 2011)

Appendix A. Hardware & Software Configurations

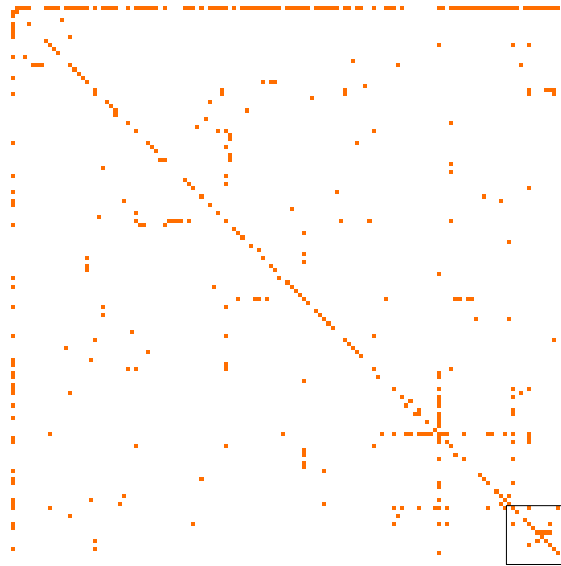


Figure A.2: Example stochastic matrix using the entire set of available signatures

This figure is similar to Fig. A.1 except using all available signatures released with the version of Snort tested. For clarity, this plot is shown using a fixed value for all events matching the temporal constraint. The bottom right box is the same as that in Fig. A.1

correlations with relatively high degree of confidence. The extent to which the predictor events cover the packet events also gives the upper bound on performance speedups when each distinct signature is an equivalence class. For the DARPA dataset, with perfect predictors this would result in at best $\frac{1049782}{13620149} = 7.7\%$ of the events being predicted and detected with a $O(1)$ computing cost (assuming equivalence classes containing only a single signature).

It is interesting to note that there are a large number of symmetries in the stochastic matrix. These symmetries account for 50% of the correlations in the testing dataset and 62% of the correlations in the corporate sample dataset. For the sample dataset the significantly higher symmetry is most likely due to the short time-frame (6 minutes) over which the sample extends. These symmetries may also represent an artifact of the SnortIDS processor or signature-sets. Further analysis is needed to determine the underlying meaning of the symmetries. The sample dataset also exhibits similar power-law event frequency distribution.

Appendix A. Hardware & Software Configurations

This implies that even for the case where the signature-set is kept constant that some gains might be achieved by utilizing trivial predictors over small sets of noisy alerts.

The relatively small proportion of alerts to packets also elucidates one of the primary performance issues with these types of detection systems and a thesis of this work. Over 90% of the information gained in using the decision procedure against incoming packet data is discarded. Since no alert fires, any features extracted (either real or potential) cannot be used for future optimizations. Each packet passes through the decision procedure. If new signatures were added to provide better predictors over the set of packets *not* associated with an alert, then the performance gains which might be achievable might span a larger portion of the dataset. Clearly, even when a dataset is highly biased, best-case performance gains using existing signature-sets is difficult to achieve.

Appendix B

Snort Configurations & Analysis

B.1 Basic Snort Configuration File

In general, the default Snort configuration (for version 2.9.1.2) was used, but with all secondary detection engines disabled or suppressed. The detection engine (with generator ID 1 was enabled for all tests). Any generator or preprocessor which was necessary for a detection signature was enabled. This included stream5, http_inspect, frag3, rpc_decode, smtp, ssh, and dcerpc2. All other generators or preprocessors which were not needed were disabled. A significant deviation from traditional testing of IDS is the introduction of a randomly generated signature-set. This was necessary in order to test the performance of the system beyond the scale and scope of available signature-sets. The statistics used to generate random signatures was generated from the official SourceFire® provided “VRT” signature-set downloaded on Oct. 17, 2011. When a traditional signature-set was used, Snort was also configured using this signature-set.

B.2 Snort Signature Profiling & Tuning

In order to produce a reasonable optimization of active signatures a signature-set needs to be assessed in the context of a given dataset. The built-in profiler for the Snort IDS provides a wealth of information which is particularly relevant to probabilistic signature activation and related techniques. An excellent guide written by Leon Ward of SourceFire Inc. provides a thorough overview of the profiling methods for tuning Snort signature-sets[91]. The following definitions reproduced directly from [91] are relevant when discussing Snort's detection engine profiler:

1. **Checks:** *The number of times rule options were checked after the **fast** pattern match process (yes, that bit is bold because it is important).*
2. **Matches:** *The number of times all rule options match, therefore traffic matching the rule has been found.*
3. **Alerts:** *The number of times the rule generated an alert. Note that this value can be different from "Matches" due to other configuration options such as alert suppression.*
4. **Microsecs:** *Total time taken processing this rule against the network traffic.*
5. **Avg/Check:** *Average time taken to check each packet against this rule.*
6. **Avg/Match:** *Average time taken to check each packet that had all options match (the rule could have generated an alert).*
7. **Avg/Nonmatch:** *Average time taken to check each packet where an event was not generated (amount of time spent checking a clean packet for bad stuff).*

In many cases tests took a long time to complete for large datasets or large signature sets. Using Snort's built-in profiling engine exposes the incredible variability in signature

Appendix B. Snort Configurations & Analysis

costs. Signature cost generally correlated with signature length. This is often due to lack of constraints on a string match or the use of complex regular expressions. For some tests costly signatures were eliminated (from both control and experimental setups) simply in order to speed up the time it took to perform a series of experiments. In other tests, high-cost signatures are used as motivation for anticipatory methods.

```

alert tcp $EXTERNAL_NET any -> $HOME_NET [80,1830]
(msg:"ORACLE Server Overflow Attempt"; flow:to_server,established;
content:"GET "; depth:4; pcre:"/^GET [\r\n]*\x3F([\r\n]*\x26)*[\x3D\r\n]{1025}/Osmi";
sid:4677; rev:6;)

```

Listing B.7: High-cost signature 4677 reproduced from a Snort IDS signature-set

Figure B.1 shows the top 10 signatures as ordered by cost-per-check (*Avg/Check*) as output by the detection engine profiler for the signature-set provided with Snort 2.9.0.3 and a test dataset (see App. A.2.) This signature-set was modified by removing all high-cost signatures except for signature 4677. This was done both to decrease the time taken to run experiments and to simplify the experimental approach to be able to show that even when abnormally costly signatures are removed that our approach is still valid. The most frequently checked, highest cost signature in the set is signature 4677. The relevant options for this signature are shown in Fig. B.7.

SID	Rev	Checks	Matches	Alerts	Microsecs	Avg/Check	Avg/Match	Avg/ Nonmatch
4677	6	123930	4	2	1.075E+09	8674.2	15763.9	8673.9
9631	5	2	0	0	8036	4018.1	0.0	4018.1
13216	5	901	0	0	2921937	3243.0	0.0	3243.0
7980	8	30	0	0	68446	2281.5	0.0	2281.5
11324	4	870	0	0	1580006	1816.1	0.0	1816.1
11620	4	2	0	0	2977	1488.7	0.0	1488.7
16067	2	81	0	0	94034	1160.9	0.0	1160.9
16301	1	431	86	59	355885	825.7	1115.4	753.5
16300	1	32	28	16	22863	714.5	816.5	0.1
17166	1	3623	0	0	1815128	501.0	0.0	501.0

Figure B.1: A Snort signature-set profile of top 10 signatures (Avg/Check)

Appendix B. Snort Configurations & Analysis

The high cost of signature 4677 is primarily due to very frequent checks (relevance) due to the packet header matching the signature's header criteria (port, protocol, and IP addresses). The high innate cost of the signature is due to the use of regular expression applied to the first 1024 bytes of the packet and use of the "O" option which disables the recursion depth limits for regular expression matches (in this case allowing any number of newlines between values up to 1024 bytes in length). The other problem with this signature is that the only other check performed is a comparison of the first 4-bytes of every qualified packet with a string to match HTTP GET requests. As a result, the regular expression match will occur for every GET request seen by the IDS. The signature is also written without use of the available HTTP preprocessor provided session state, but this may be intentional as it may be that the attack can be performed irrespective of the state of the HTTP connection.

The traditional tuning methodology would suggest that signature 4677 be modified to reduce its cost or simply removed from the signature-set due to its high cost. However, modification of a signature may not be possible due to the skill required or due to the innate requirements of detecting a particular event (perhaps requiring high-cost regular expressions or exhaustive search of a large portion of individual packets).

In cases where a signature cannot be easily modified and acceptable performance cannot be achieved, removal of a signature is currently the only available option. This may be acceptable when the event is not relevant for the network, target hosts, or targeted services, but it guarantees that the associated events will be missed. It is not clear from Fig. B.1, but signature 4677 accounts for the vast majority of the computing cost of the signature-set. The signature performs terribly and should certainly be disabled if it cannot be modified. However, for purposes of argument, we will use this signature in particular to show experimentally that abnormally expensive signatures do not necessarily need to be removed in order to achieve acceptable performance. We also examine experimental evidence that even when such signatures are disabled there are still optimizations based on

Appendix B. Snort Configurations & Analysis

probabilistic signature activation which can improve performance if packets are still being lost.

B.2.1 0min-10min Signature Profiles

SID	GID	Rev	Checks	Matches	Alerts	Microsecs	Avg/ Check	Avg/ Match	Avg/ Nonmatch
8701	1	2	151173	0	0	7260819	48.0	0.0	48.0
17512	1	3	106341	0	0	6163985	58.0	0.0	58.0
14990	1	1	52365	0	0	4898394	93.5	0.0	93.5
2707	1	3	3525113	1255	201	4128901	1.2	7.6	1.2
16014	1	2	149959	0	0	3290514	21.9	0.0	21.9
17513	1	3	201014	0	0	3082106	15.3	0.0	15.3
13216	1	5	903	0	0	2913315	3226.3	0.0	3226.3
12183	1	3	2978094	0	0	2187487	0.7	0.0	0.7
17166	1	1	3621	0	0	1830787	505.6	0.0	505.6
Total			93,446,968			78,541,173			

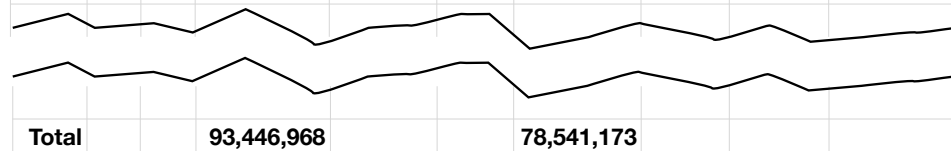


Table B.1: Top 10 signatures in terms of total CPU-time in microseconds

B.2.2 0min-10min Cost Policy Signature Profile

We can run the same profiler against a signature-set which is using a set of activation probabilities for high-cost signatures. The total CPU-time used for all signatures with the activation probabilities for Test 38-2K is 67.983 seconds, a modest 13% decrease from 78.541 seconds of CPU-time. Oddly, when directly comparing both checks and CPU-time between the control setup and the experimental setup, we note in Table B.3 that there is a significant decrease in CPU-time but a larger number of signature checks. Interestingly the number of checks was increased by 1.5 million times. It is unclear why, but could be related to the use of new flowbits for the signatures.

Appendix B. Snort Configurations & Analysis

SID	GID	Rev	Checks	Matches	Alerts	Microsecs	Avg/ Check	Avg/ Match	Avg/ Nonmatch
2707	1	3	4655669	202	190	3578613	0.8	9.8	0.8
623	1	7	9078531	0	0	3019473	0.3	0	0.3
13216	1	5	1169	0	0	2679802	2292.4	0	2292.4
14990	1	1	65770	0	0	2550712	38.8	0	38.8
17513	1	3	171454	0	0	2524635	14.7	0	14.7
16014	1	2	174697	0	0	2418601	13.8	0	13.8
12183	1	3	3036831	0	0	2414838	0.8	0	0.8
17512	1	3	54721	0	0	2256843	41.2	0	41.2
624	1	8	9183604	0	0	2127341	0.2	0	0.2
Total			96,704,555			67,983,590			

Table B.2: Top 10 signatures in terms of total CPU-time using a Cost policy

SID	No Optimization		Cost Policy		Difference	
	Checks	Microseconds	Check	Microseconds	Checks	Microseconds
8701	151173	7260819	208596	1062796	57423	-6198023
17512	106341	6163985	54721	2256843	-51620	-3907142
14990	52365	4898394	65770	2550712	13405	-2347682
2707	3525113	4128901	4655669	3578613	1130556	-550288
16014	149959	3290514	174697	2418601	24738	-871913
17513	201014	3082106	171454	2524635	-29560	-557471
13216	903	2913315	1169	2679802	266	-233513
12183	2978094	2187487	3036831	2414838	58737	227351
17166	3621	1830787	4295	1740004	674	-90783
630	8947581	1785615	9334371	1907309	386790	121694
Totals	16,116,164	37,541,923	17,707,573	23,134,153	1,591,409	-14,407,770

Table B.3: Cost policy performance differences in total CPU-time and checks

B.3 Custom Snort Signatures

B.3.1 Detection Rules

HTTP Detection

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any
(msg:"HTTP Protocol Detect GET";
flow:to_server,established;
content:"GET"; depth:5;
sid:2000001; rev:1;)
```

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any
(msg:"HTTP Protocol Detect POST";
flow:to_server,established;
content:"POST"; depth:6;
sid:2000002; rev:1;)
```

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any
(msg:"HTTP Protocol Detect HEAD";
flow:to_server,established;
content:"HEAD"; depth:6;
sid:2000003; rev:1;)
```

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any
(msg:"HTTP Protocol Detect HTTP Response";
flow:to_server,established;
content:!"GET"; depth:5;
content:!"POST"; depth:6;
content:!"HEAD"; depth:6;
content:"HTTP"; depth:100;
sid:2000004; rev:1;)
```

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any
(msg:"HTTP Content Detect HTTP URI";
flow:to_server,established;
content:"http://"; nocase;
sid:2000005; rev:1;)
```

Listing B.8: Simple HTTP activity detection signatures

Appendix B. Snort Configurations & Analysis

TCP Detection

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any
  (msg:"Alert on any TCP packet
  flow:to_client,established";
  flow:to_client,established;
  flowbits:set,checked.2090001;
  sid:2090001; rev:1;)
```

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any
  (msg:"Alert on any TCP packet
  flow:to_server,established";
  flow:to_server,established;
  flowbits:set,checked.2091002;
  sid:2091002; rev:1;)
```

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any
  (msg:"Alert on any TCP packet
  flow:established";
  flow:established;
  flowbits:set,checked.2092003;
  sid:2092003; rev:1;)
```

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any
  (msg:"Alert on any TCP packet";
  flowbits:set,checked.2092004;
  sid:2092004; rev:1;)
```

Listing B.9: TCP Detection Signatures

Appendix C

Issues & Confounding Factors

C.1 Event Suppression

One issue with live configurations is that many potential events may be suppressed through Snort's event processing tuning mechanisms (i.e. "detection", "rate", and "event" filters and event "suppression")[79]. Thresholding is highly data and load dependent. A high-rate IP sequence that results in duplicate events in a small time window will result in high filtering rates of output events. The same network load but with a different alert may not result in any filtering. If Snort is configured using default thresholding and event suppression, then live traffic processing coverage may be less than the coverage measured in an offline test. However, since the filtering occurs at the output of the IDS, each packet still sees the entire signature-set, and gains via anticipatory bias are still relevant. Further, if coverage is measured on a live system, this would result in an underestimate of packet coverage by some constant factor and thus an underestimate of any performance gains to be had.

Interestingly, packet coverage also appears to follow a power-law distribution in respect to event type. A relatively small number of event types account for a significant portion of

qualified events. This is potentially advantageous for a predictive system as is shown in the next chapter based on an analysis of the performance of the Snort IDS on a test system.

C.2 Packet Loss

Packet losses only occur when hardware or software buffers fill and data can no longer be stored while it is waiting to be processed. In the case of a network IDS, the hardware buffer resides on the network interface card. In the NAPI, if the userspace Snort process is unable to retrieve a packet in time, it is overwritten in the network interface card's incoming ring buffer. Because there is no userspace buffering or caching of network traffic, the packet losses incurred are due to CPU contention between the thread which is performing IDS detection and the thread which is copying packets from the network interface card. In some cases, buffer sizes can be expanded to accommodate packet loss issues. However, in the experiments being performed, packet loss will eventually occur not matter how large the buffers are made as signature-set size is increased.

In order to test packet loss two separate experiments were performed: line saturation, and line capacity. In initial experiments, packets were transmitted from the same machine as they were being received. This was shown to have little impact on the scaling performance. When packets were transmitted between two separate machines the performance characteristics were very similar. In all two-machine experiments the secondary machine was an Intel i7 Macbook Pro (as described in Appendix A.1). Packets were transmitted from the network machine using the open source *tcpreplay* application.

In all scaling performance experiments a single Snort process is configured to listen on the connected interface device and to log packet-processing statistics once per second. The signature-set size was varied from 1, 000 to 40, 000 in increments of 10, 000 signatures with 10 trials per signature-set size. The sending machine was configured to delay packet

Appendix C. Issues & Confounding Factors

transmission according to the expected startup time for the Snort IDS (see Sect.4.2.1). Two independent sets of data were run to determine if the dataset would play a role in performance (see Appendix A.2). See Appendix A.1 for information about tcpreplay configuration options.

The line saturation test sends the packets as fast as possible without dropping packets upon transmission using the *-t* (*-topspeed*) option of tcpreplay. At the receiving end, packets are processed normally by a single Snort process listening to the connected interface device configured in *promiscuous* mode (forwards all packets regardless of source or destination).

C.2.1 I/O Costs from Alert Output

The use of randomly generated signatures resulted in a concern of whether I/O costs would play a role. It was expected that the randomly generated signatures may produce a greater number of alerts and consume a significant amount of CPU-time writing alert events to disk. A single experiment was run on the test platform which disabled all alert output. The slopes of the respective CPU-time scaling curves differed by about 5%, from 103634 (no alert output) to 109755 (with alert output). However, the number of alerts produced by the random signature-set is actually significantly less than the default signature-set for the same dataset. This characteristic actually serves to decrease the apparent per-signature cost. As was learned from experiments with high-coverage signatures, I/O costs can play a significant role in system performance when the portion of packets resulting in event output is large. This finding is demonstrated in Chapt. 6.2 and is the basis for the use of Probabilistic Signature Activation as described in Chapt. 3.2.

C.2.2 I/O Costs from Packet Replay

All of the scaling performance tests were run on a single machine with one process transmitting data on a virtual interface and a separate IDS process receiving data on the same virtual interface¹. This was done primarily to eliminate issues on the network in which a random subset of packets would be dropped prior to being seen by the IDS. It was important to measure the IDS dropping packets and not the hardware in between the sending machine and the receiving user process. The use of a virtual interface means that packets are buffered in kernel memory.

¹<http://vtun.sourceforge.net/tun/> - *TunTap Project* (Source Code - Retrieved Aug. 12, 2012)

Appendix D

Scripts & Algorithms

D.1 Packet Wrangler

Listing D.10: Packet Wrangler: predictEquivalenceClasses(Alert Set A_i , Connection C_i)

Require: Set E of equivalence classes ξ

Require: Alert Set A_i which is to be mapped into likely equivalence classes

Require: A stochastic matrix T determining likely future equivalence classes for each alert

Calculate anticipated set of equivalence classes $E_i = \{(\xi, P(\xi) \in E)\}$ using T

Call `updateFilters(C_i, E_i)`

Listing D.11: Packet Wrangler: updateAllFilters

Require: Set E of equivalence classes ξ

Require: k instances of Snort and an input filter for each ξ

Require: ξ_0 defined as the global equivalence class (union set of equivalence classes)

Require: hashmap H mapping each ξ to relevant connections (Address1 \leftrightarrow Address2)

for all $\xi \in E$ **do**

$ConnectionList$ = list of connections associated with ξ

 Remove all filter expressions for $Snort_\xi$

for all $Connection \in ConnectionList$ **do**

$f \leftarrow$ filter expression for $Connection$

 Add f to filter set for $Snort_\xi$

 Remove f from filter set for $Snort_{\xi_0}$ (filter set for global ξ)

end for

end for

Ensure: Each $Snort_i$ is running with input filters for connections associated with ξ_i

Appendix D. Scripts & Algorithms

Listing D.12: Packet Wrangler: updateFilter(Connection C_i , Equivalence classes E_i)

```
Require: Set  $E$  of equivalence classes  $\xi$ 
Require: An instance of Snort and associated input filter running for each  $\xi$ 
Require:  $\xi_0$  defined as the global equivalence class (union set of equivalence classes)
Require: hashmap  $G$  mapping each connection to previous set of  $\xi$ 
Require: hashmap  $H$  mapping each connection to a set of anticipated  $\xi$ 
Require: A connection  $C_i$ 
Require: A set of anticipated equivalence classes  $E_i$ 
   $F \leftarrow G(C_i)$ , set of all equivalence classes previously associated with  $C_i$ 
  for all  $\xi \in F$  do
    Remove filter for  $C_i$  from IDS instance  $Snort_\xi$ 
  end for
  if  $H$  is empty then
    Add filter for  $C_i$  to  $Snort_0$ 
    Return
  end if
  Remove filter for  $C_i$  from  $Snort_0$ 
  for all  $\xi \in H$  do
    Add filter for  $C_i$  to IDS instance  $Snort_\xi$ 
  end for
Ensure: Each  $Snort_i$  is running with input filters for connections associated with  $\xi_i$ 
```

D.2 Gain Curve Calculations

In respect to anticipation the usable coverage of a particular signature-set is proportional to the total number of packets which produce alerts and the number of these which belong to the smaller equivalence class, A_p and A_r . Although it is possible for a single event to make multiple predictions, we currently only care about the minimum we might be able to achieve. If we assume that each event can only be used once for a single prediction, then we can get an upper bound on our gain given that each event is used for prediction at most once. Written in Mathematica, Listing D.13 shows the setup and listings D.14 and D.15 show evaluation for calculating a family of constant gain curves. It is important to reiterate that the curves are particular to the computing system signature-set. Many different factors can play a role in scaling performance specifics and the resulting gain estimates are only

Appendix D. Scripts & Algorithms

representative examples and may not apply to different architectures, detection systems, or configurations.

Listing D.13: Setup for gain curve computation in Mathematica

```
Ar[Ap_, Aq_] := Ap*Aq
Ctime[n_] := 103634 n + 4.01621 10^9
Closs[n_] := 0.9417 - 1.0392^(-0.008 n)
Cost[n_] := Ctime[n]/Max[0, (1 - Closs[n])];
Cprimary[Ar_, n_, k_] := Cost[n]
Csecondary[Ar_, n_, k_] := Cost[k] Ar
Cbootstrap[Ar_, n_, k_] := Cost[n] (1 - Ar)
Cdelta[Ar_, n_, k_] := Cprimary[Ar, n, k] - (Csecondary[Ar, n, k] + Cbootstrap[Ar, n, k])
G[Ar_, n_, k_] := Cprimary[Ar, n, k]/(Csecondary[Ar, n, k] + Cbootstrap[Ar, n, k]) - 1
Gtime[Ar_, n_, k_] := Ctime[n]/(Ctime[n] (1 - Ar) + (Ctime[k] Ar)) - 1
Gloss[Ar_, n_, k_] := Closs[n]/(Closs[n] (1 - Ar) + (Closs[k] Ar)) - 1
```

$C_{primary}$ is the cost of running the primary IDS without any type of forwarding or anticipatory method. $C_{bootstrap}$ is the primary instance but with traffic shunted to a secondary instance based on predictions. $C_{secondary}$ is the secondary IDS instance which processes only forwarded traffic with a smaller signature-set.

Listing D.14: Gain curves computation for $n=4320$, G in $\{1,10,20,\dots,100\}$

```
Evaluate[Plot[
  Table[FindRoot[G[Ar, 4320, k]*100 - Overhead, {Ar, 0.216}][[1,2]],
    {Overhead, 1, 100, 10}],
  {k, 1, 4300},
  PlotRange -> {0, 1},
  AxesLabel -> {"k", "Ar"},
  PlotLabel -> StringJoin["Ar(k) for G in {1,10,20,...,100}%", n=4320"]
]]
```

Listing D.15: Gain curves computation for $n=100000$, G in $\{1,10,20,\dots,100\}$

```
Evaluate[Plot[
  Table[FindRoot[G[Ar, 100000, k]*100 - Overhead, {Ar, 0.216}][[1,2]],
    {Overhead, 1, 1000, 100}],
  {k, 1, 100000 - 100},
```

Appendix D. Scripts & Algorithms

```
PlotRange -> {0, 1},  
AxesLabel -> {"k", "Ar"},  
PlotLabel -> StringJoin["Ar(k) for G in {1,100,200,...,1000}%, n=100,000"]  
]]
```

References

- [1] T Abraham, *IDDM: Intrusion detection using data mining techniques*, Tech. report, Defense Science Technology Organisation, January 2001.
- [2] A Agah, S.K Das, K Basu, and M Asadi, *Intrusion detection in sensor networks: a non-cooperative game approach*, Proceedings of Third IEEE International Symposium on Network Computing and Applications (NCA), 2004, pp. 343–346.
- [3] Alfred V Aho and Margaret J Corasick, *Efficient string matching: an aid to bibliographic search*, Communications of the ACM **18** (1975), no. 6, 333–340.
- [4] E Albin and N C Rowe, *A Realistic Experimental Comparison of the Suricata and Snort Intrusion-Detection Systems*, Proceedings of 26th International Conference on Advanced Information Networking and Applications Workshops (WAINA), IEEE Computer Society, 2012, pp. 122–127.
- [5] Adeeb Alhomoud, Rashid Munir, Jules Pagna Disso, Irfan Awan, and A Al-Dhelaan, *Performance Evaluation Study of Intrusion Detection Systems*, Procedia Computer Science **5** (2011), 173–180 (English).
- [6] J Allen, A Christie, W Fithen, J McHugh, J Pickel, and Ed Stoner, *State of the practice of intrusion detection technologies*, Networked Systems Survivability Program, Technical Report CMU/SEI-99-TR-028 (2000), 1–219.
- [7] T Alpcan and T Basar, *A game theoretic approach to decision and analysis in network intrusion detection*, Proceedings of 42nd IEEE Conference on Decision and Control, 2003, pp. 2595–2600.
- [8] N Amor, S Benferhat, and Z Elouedi, *Naive bayes vs decision trees in intrusion detection systems*, Proceedings of the 2004 ACM symposium on Applied Computing (2004), 420–424.

References

- [9] Anderson, D Engelhardt, and D Marriott, *Event handling system*, US Patent Office (2003), no. 12/004,980.
- [10] M Anderson and D Engelhardt, *Data processing architecture*, WIPO (2002), no. WO 02/088988AI.
- [11] Mark Anderson, Dean Engelhardt, Damian Marriott, and Suneel Randhawa, *Data view of a modelling system*, US Patent Office (2006), no. 7,027,055 B2.
- [12] Mark Anderson, Dean Engelhardt, Damion Marriott, and Suneel Randhawa, *Geographic View of a Modelling System*, US Patent Office (2009), no. 7,250,944 B2, 1–67.
- [13] Mark Sephen Anderson, Dean Engelhardt, Damian Marriott, and Singh Randhawa, *Data processing and observation system*, US Patent Office (2006), no. 7,085,683.
- [14] S Axelsson, *Research in Intrusion Detection Systems: A Survey*, Tech. Report 98-17, Chalmers University of Technology, 1999.
- [15] ———, *Intrusion detection systems: A survey and taxonomy*, Depart. of Computer Engineering (2000), 1–27.
- [16] ———, *The base-rate fallacy and the difficulty of intrusion detection*, ACM Transactions on Information and System Security (2000), 186–205.
- [17] I Balepin, S Maltsev, J Rowe, and K Levitt, *Using specification-based intrusion detection for automated response*, Proceedings of Recent Advances in Intrusion Detection (2003), 136–154.
- [18] Pere Barlet-Ros, Gianluca Iannaccone, Josep Sanjuà-Cuxart, and Josep Solé-Pareta, *Predictive resource management of multiple monitoring applications*, IEEE/ACM Transactions on Networking (TON) **19** (2011), no. 3, 788–801.
- [19] Matt Bishop, *Computer Security: Art and Science*, Addison-Wesley, January 2003.
- [20] C Brown, A Cowperthwaite, A Hijazi, and A Somayaji, *Analysis of the 1999 DARPA/Lincoln Laboratory IDS evaluation data with NetADHICT*, Proceedings of IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA) (2009), 1–7.
- [21] ST Brugger and Jedediah Chow, *An assessment of the DARPA IDS Evaluation Dataset using Snort*, Tech. report, UCDAVIS Department of Computer Science, November 2005.

References

- [22] Terry Brugger, *KDD Cup '99 dataset (Network Intrusion) considered harmful*, KDNuggets News **18** (2007), no. 4, 1–2.
- [23] D Brumley, J Newsome, and D Song, *Theory and techniques for automatic generation of vulnerability-based signatures*, IEEE Transactions on Dependable and Secure Computing **5** (2008), no. 4, 224–241.
- [24] D Brumley, J Newsome, D Song, Hao Wang, and S Jha, *Towards Automatic Generation of Vulnerability-Based Signatures*, Proceedings of IEEE Symposium on Security and Privacy (S&P'06), IEEE, January 2006, pp. 2–16.
- [25] J Caballero, Z Liang, P Poosankam, and D Song, *Towards generating high coverage vulnerability-based signatures with protocol-level constraint-guided exploration*, Proceedings of Recent Advances in Intrusion Detection (2009), 161–181.
- [26] J Cannady and J Harrell, *A comparative analysis of current intrusion detection technologies*, Proceedings of Technology in Information Security Conference (1996), 212–218.
- [27] D Dasgupta and F Gonzalez, *An intelligent decision support system for intrusion detection and response*, Proceedings of the International Workshop on Mathematical Methods, Models and Architectures for Computer Networks Security (2001), 1–14.
- [28] D Day and B Burns, *A performance analysis of snort and suricata network intrusion detection and prevention engines*, Proceedings of The Fifth International Conference on Digital Society (ICDS) (2011), 187–192.
- [29] H Debar, M Dacier, and A Wespi, *Towards a taxonomy of intrusion-detection systems*, Computer Networks **31** (1999), no. 8, 805–822.
- [30] ———, *A revised taxonomy for intrusion-detection systems*, Annals of Telecommunications **55** (2000), no. 7, 361–378.
- [31] D Denning, *An intrusion-detection model*, IEEE Transactions on Software Engineering **SE-13** (1987), no. 2, 222–232.
- [32] L Deri, *High-speed dynamic packet filtering*, Journal of Network and Systems Management **15** (2007), no. 3, 401–415.
- [33] Vassilis Dimopoulos, Ioannis Papaefstathiou, and Dionisios Pnevmatikatos, *A Memory-Efficient Reconfigurable Aho-Corasick FSM Implementation for Intrusion Detection Systems*, Proceedings of International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation, IEEE, 2007, pp. 186–193.

References

- [34] H Dreger, A Feldmann, V Paxson, and Robin Sommer, *Predicting the resource consumption of network intrusion detection systems*, Proceedings of Recent Advances in Intrusion Detection, January 2008, pp. 135–154.
- [35] I Dubrawsky and R Saville, *SAFE: IDS Deployment, Tuning, and Logging in Depth* Authors, CISCO SAFE Whitepaper (2003), 1–58.
- [36] F C Eigler, V Prasad, W Cohen, H Nguyen, M Hunt, Jim Keniston, and Brad Chen, *Architecture of systemtap: a Linux trace/probe tool*, Tech. report, RedHat, 2005.
- [37] D Engelhardt and M Anderson, *A distributed multi-agent architecture for computer security situational awareness*, Proceedings of Sixth International Conference of Information Fusion **1** (2003), 1–8.
- [38] Vegard Engen, Jonathan Vincent, and Keith Phalp, *Exploring discrepancies in findings obtained with the KDD Cup '99 data set*, Intelligent Data Analysis **15** (2011), no. 2, 251–276.
- [39] W Fan, W Lee, S Stolfo, and M Miller, *A multiple model cost-sensitive approach for intrusion detection*, Lecture Notes in Computer Science (LNCS) **1810** (2000), 142–154.
- [40] Stephane Forrest, Steven A Hofmeyr, and Anil Somayaji, *Computer immunology*, Communications of the ACM **40** (1997), no. 10, 88–96.
- [41] J Frank, *Artificial intelligence and intrusion detection: Current and future directions*, Proceedings of the 17th National Computer Security Conference (1994), 1–12.
- [42] J Friedman, R Kohavi, and Y Yun, *Lazy decision trees*, Proceedings of the Association for the Advancement of Artificial Intelligence (1996), 1–8.
- [43] Sunny Fugate, *Visual Representations of Flow Data and the Value of Visual Language*, FloCon 2008, January 2008.
- [44] C Geib and R Goldman, *Plan recognition in intrusion detection systems*, DARPA Information Survivability Conference (2001), 1–10.
- [45] J Haines, R Lippmann, D Fried, and M Zissman, *1999 DARPA intrusion detection evaluation: Design and procedures*, Tech. report, MIT, January 2001.
- [46] J Hellerstein and F Zhang, *A statistical approach to predictive detection*, Computer Networks (2001), 77–95.
- [47] A Herzog, N Shahmehri, and C Duma, *An ontology of information security*, International Journal of Information Security and Privacy **1** (2007), no. 4, 1–23.

References

- [48] K Jackson, *Intrusion detection system (IDS) product survey*, Tech. report, Los Alamos National Laboratory, January 1999.
- [49] P Kabiri and A A Ghorbani, *Research on intrusion detection and response: A survey*, International Journal of Network Security **1** (2005), no. 2, 84–102.
- [50] Rohana J Karunamuni and Shunpu Zhang, *Empirical Bayes detection of a change in distribution*, Annals of the Institute of Statistical Mathematics **48** (1996), no. 2, 229–246 (English).
- [51] RA Kemmerer and Giovanni Vigna, *Intrusion detection: a brief history and overview*, Computer **35** (2002), no. 4, 27–30.
- [52] TM Khoshgoftaar, *Predicting software development errors using software complexity metrics*, IEEE Journal on Selected Areas in Communications **8** (1990), no. 2, 253–261.
- [53] C Kruegel and T Toth, *Using decision trees to improve signature-based intrusion detection*, Lecture Notes in Computer Science (LNCS) **2820** (2003), 173–191.
- [54] S Kumar, *Classification and detection of computer intrusions*, Purdue University (1995), 1–165.
- [55] LJ LaPadula, *Compendium of anomaly detection and reaction tools and projects*, Tech. Report MP 99B0000018R1, MITRE, January 2000.
- [56] W Lee, W Fan, M Miller, SJ Stolfo, and E Zadok, *Toward cost-sensitive modeling for intrusion detection and response*, Journal of Computer Security **10** (2002), 5–22.
- [57] W Lee, SJ Solfo, and Kui W Mok, *Adaptive intrusion detection: A data mining approach*, Artificial Intelligence Review **14** (2000), 553–567.
- [58] Wenke Lee, João B D Cabrera, Ashley Thomas, Niranjan Balwalli, Sunmeet Saluja, and Yi Zhang, *Performance adaptation in real-time intrusion detection systems*, Lecture Notes in Computer Science (LNCS) **2516** (2002), no. Chapter 14, 252–273.
- [59] K Levitt, *Intrusion detection: current capabilities and future directions*, Proceedings of the 18th Annual Computer Security Applications Conference (2003), 365–367.
- [60] X Li, *A scalable decision tree system and its application in pattern recognition and intrusion detection*, Decision Support Systems **41** (2005), no. 1, 112–130.
- [61] X Li and N Ye, *Decision tree classifiers for computer intrusion detection*, Parallel and Distributed Computing Practices **4** (2001), no. 2, 179–190.

References

- [62] R Lippmann, D Fried, I Graf, J W Haines, K R Kendall, D McClung, D Weber, D Wyschogrod, R K Cunningham, and M A Zissman, *Evaluating intrusion detection systems: The 1998 DARPA off-line intrusion detection evaluation*, DARPA Information Survivability Conference and Exposition **2** (2000), 12–26.
- [63] R Lippmann, J Haines, D Fried, J Korba, and K Das, *The 1999 DARPA off-line intrusion detection evaluation*, *Computer Networks* (2000), 579–595.
- [64] Y. Liu, H. Man, and C. Comaniciu, *A game theoretic approach to efficient mixed strategies for intrusion detection*, *Proceedings of the IEEE International Conference on Communications (ICC)* **5** (2006), 2201–2206.
- [65] E Lundin and E Jonsson, *Survey of intrusion detection research*, Tech. report, Chalmers University of Technology, January 2002.
- [66] T F Lunt, A Tamaru, F Gilham, R Jagannathan, C Jalali, P G Neumann, H S Javitz, A Valdes, and T D Garvey, *A real time intrusion detection expert system (IDES)*, Tech. Report 6784, SRI International, 1990.
- [67] M V Mahoney and P K Chan, *An analysis of the 1999 DARPA/Lincoln Laboratory evaluation data for network anomaly detection*, *Proceedings of Recent Advances in Intrusion Detection*, January 2003, pp. 220–237.
- [68] Marc Norton, *Optimizing Pattern Matching for Intrusion Detection*, Tech. report, SourceFire Inc, June 2004.
- [69] Lies Notebaert, Geert Crombez, Stefaan Van Damme, Jan De Houwer, and Jan Theeuwes, *Signals of threat do not capture, but prioritize, attention: A conditioning approach.*, *Emotion* **11** (2011), no. 1, 81–89 (English).
- [70] J Novak and Steve Sturges, *Target-Based TCP Stream Reassembly*, Tech. report, SourceFire Inc, 2007.
- [71] A. Papadogiannakis, M Polychronakis, and E.P. Markatos, *Improving the accuracy of network intrusion detection systems under load using selective packet discarding*, *Proceedings of the Third European Workshop on System Security* (2010), 15–21.
- [72] Vern Paxson, *Bro: A System for Detecting Network Intruders in Real-Time*, *Computer Networks: The International Journal of Computer and Telecommunications* **31** (1999), no. 23-24, 2435–2463.
- [73] S Peddabachigari, A Abraham, and J Thomas, *Intrusion detection systems using decision trees and support vector machines*, *International Journal of Applied Science and Computations* (2004), 118–134.

References

- [74] M Pollak, *Optimal Detection of a Change in Distribution*, The Annals of Statistics **13** (1985), no. 1, 206–227.
- [75] J Quinlan, *Discovering rules by induction from large collections of examples*, Expert Systems in the Micro-electronic Age (1979), 168–201.
- [76] ———, *Induction of decision trees*, Machine Learning **1** (1986), 81–106.
- [77] ———, *Learning logical definitions from relations*, Machine Learning **5** (1990), 239–266.
- [78] M Roesch, *Snort-lightweight intrusion detection for networks*, Proceedings of the 13th USENIX Systems Administrators Conference (LISA) (1999), 229–238.
- [79] Martin Roesch and Chris Green, *SNORT Users Manual 2.8.6*, Tech. report, SourceFire Inc, April 2010.
- [80] Sebastian Roschke, Feng Cheng, and Christoph Meinel, *A new alert correlation algorithm based on attack graph*, Proceedings of the 4th international conference on Computational Intelligence in Security for Information Systems (CISIS), Springer-Verlag, June 2011, pp. 58–67.
- [81] K Salah and A Kahtani, *Improving snort performance under linux*, Communications, IET **3** (2009), no. 12, 1883–1895.
- [82] Nabil Schear, David R Albrecht, and Nikita Borisov, *High-Speed Matching of Vulnerability Signatures*, Proceedings of Recent Advances in Intrusion Detection (Richard Lippmann, Engin Kirda, and Ari Trachtenberg, eds.), Springer Berlin Heidelberg, 2008, pp. 155–174.
- [83] V Shen, T Yu, SM Thebaut, and L R Paulsen, *Identifying error-prone software—an empirical study*, IEEE Transactions on Software Engineering **SE-11** (1985), no. 4, 317–324.
- [84] Steven R Snapp, James Brentano, T L Goan, T Grance, L Heberlein, C Ho, KN Levitt, B Mukherjee, D Mansur, K Pon, and S E Smaha, *Intrusion Detection Systems (IDS): A Survey of Existing Systems and a Proposed Distributed IDS Architecture*, UC Davis, Report No. CSE-91-7 (1991), 1–18.
- [85] Natalia Stakhanova, Samik Basu, and Johnny Wong, *A taxonomy of intrusion response systems*, International Journal of Information and Computer Security **1** (2007), no. 1/2, 168–184.

References

- [86] C Strasburg, N Stakhanova, S Basu, and JS Wong, *Intrusion response cost assessment methodology*, Proceedings of the 4th International Symposium on Information, Computer, and Communications Security (2009), 388–391.
- [87] H Teng, K Chen, and Stephen Lu, *Adaptive real-time anomaly detection using inductively generated sequential patterns*, Proceedings of the 1990 IEEE Symposium on Research in Security and Privacy (1990), 278–284.
- [88] Tung Tran, Issam Aib, Ehab Al-Shaer, and Raouf Boutaba, *An evasive attack on SNORT flowbits*, Proceedings of Network Operations and Management Symposium (NOMS) (2012), 351–358.
- [89] T Verwoerd and R Hunt, *Intrusion detection techniques and approaches*, Computer Communications **25** (2002), 1356–1365.
- [90] L Vespa, M Mathew, and N Weng, *Predictive Pattern Matching for Scalable Network Intrusion Detection*, Lecture Notes In Computer Science: Information and Communications Security **5927** (2009), 254–267.
- [91] Leon Ward, *Improving your custom Snort rules*, Tech. report, Sourcefire, November 2011.
- [92] Zhenwei Yu, Jeffrey Tsai, and Thomas Weigert, *An adaptive automatically tuning intrusion detection system*, ACM Transactions on Autonomous and Adaptive Systems (TAAS) **3** (2008), no. 3, 10:1–10:25.
- [93] Y Zhang, X Fan, and Y Wang, *Attack grammar: A new approach to modeling and analyzing network attack sequences*, Proceedings of Annual Computer Security Applications Conference (2008), 215–224.
- [94] B Zhu and A Ghorbani, *Alert correlation for extracting attack strategies*, International Journal of Network Security (2006), 1–28.