POLITECNICO
MILANO

University of California
Department of Computer Science

DOTTORATO DI RICERCA IN INGEGNERIA
DELL'INFORMAZIONE

Integrated Detection of Anomalous Behavior of Computer Infrastructures

Doctoral Dissertation of:
Federico Maggi

Advisor:
Prof. Stefano Zanero

Tutor:
Prof. Letizia Tanca

Supervisor of the Doctoral Program:
Prof. Patrizio Colaneri

December 2013

Preface

This thesis embraces all the efforts that I put during the last three years as a PhD student at Politecnico di Milano. I have been working under the supervision of Prof. S. Zanero and Prof. G. Serazzi, who is also the leader of the research group I am part of. In this time frame I had the wonderful opportunity of being "initiated" to research, which radically changed the way I look at things: I found my natural *"thinking outside the box"* attitude — that was probably well-hidden under a thick layer of lack-of-opportunities, I took part of very interesting joint works — among which the year I spent at the Computer Security Laboratory at UC Santa Barbara is at the first place, and I discovered the Zen of my life.

My research is all about *computers* and every other technology possibly related to them. Clearly, the way I look at computers has changed a bit since when I was seven. Still, I can remember me, typing on that 64 in front of a tube TV screen, trying to get that d—n routine written in to work. I was just playing, obviously, but when I recently found a picture of me in front of that screen...it all became clear.

So, although my attempt of writing a program to authenticate myself was a little bit naive — being limited to a print instruction up to that point apart, of course — I thought *"maybe I am not in the wrong place, and the fact that my research is still about security is a good sign"*!

Many years later, this work comes to life. There is a humongous amount of people that, directly or indirectly, have contributed to my research and, in particular, to this work. Since my first step into the lab, I will not, ever, be thankful enough to Stefano, who, despite my skepticism, convinced me to submit that application for the PhD program. For trusting me since the very first moment I am thankful to Prof. G. Serazzi as well, who has been always supportive. For hosting and supporting my research abroad I thank Prof. G. Vigna, Prof. C. Kruegel, and Prof. R. Kemmerer. Also, I wish to thank Prof. M. Matteucci for the great collaboration, Prof. I. Epifani for her insightful suggestions and Prof. H. Bos for the detailed review and the constructive comments.

On the colleagues-side of this acknowledgments I put all the fellows of Room 157, Guido, the crew of the seclab and, in particular, Wil with whom I shared all the pain of paper writing between Sept '08 and Jun '09.

On the friends-side of this list Lorenzo and Simona go first, for being our family.

I have tried to translate in simple words the infinite gratitude I have and will always have to Valentina and my parents for being my fixed point in my life. Obviously, I failed.

Federico Maggi
Milano
September 2009

Abstract

This dissertation details our research on anomaly detection techniques, that are central to several classic security-related tasks such as network monitoring, but it also have broader applications such as program behavior characterization or malware classification. In particular, we worked on anomaly detection from three different perspective, with the common goal of recognizing awkward activity on computer infrastructures. In fact, a computer system has several weak spots that must be protected to avoid attackers to take advantage of them. We focused on protecting the operating system, central to any computer, to avoid malicious code to subvert its normal activity. Secondly, we concentrated on protecting the web applications, which can be considered the modern, shared operating systems; because of their immense popularity, they have indeed become the most targeted entry point to violate a system. Last, we experimented with novel techniques with the aim of identifying related events (e.g., alerts reported by intrusion detection systems) to build new and more compact knowledge to detect malicious activity on large-scale systems.

Our contributions regarding host-based protection systems focus on characterizing a process' behavior through the system calls invoked into the kernel. In particular, we engineered and carefully tested different versions of a multi-model detection system using both stochastic and deterministic models to capture the features of the system calls during normal operation of the operating system. Besides demonstrating the effectiveness of our approaches, we confirmed that the use of finite-state, deterministic models allow to detect deviations from the process' control flow with the highest accuracy; however, our contribution combine this effectiveness with advanced models for the system calls' arguments resulting in a significantly decreased number of false alarms.

Our contributions regarding web-based protection systems focus on advanced training procedures to enable learning systems to perform well even in presence of changes in the web application source code — particularly frequent in the Web 2.0 era. We also addressed data scarcity issues that is a real problem when deploying an anomaly detector to protect a new, never-used-before application. Both these issues dramatically decrease the detection capabilities of an intrusion detection system but can be effectively mitigated by adopting the techniques we propose.

Last, we investigated the use of different stochastic and fuzzy models to perform automatic alert correlation, which is as post processing step to intrusion detection. We proposed a fuzzy model that formally defines the errors that inevitably occur if time-based alert aggregation (i.e., two alerts are considered correlated if they are close in time) is used. This model allow to account for measurements errors and avoid false correlations due to delays, for instance, or incorrect parameter settings. In addition, we defined a model to describe the alert generation as a stochastic process and experimented with non-parametric statistical tests to define robust, zero-configuration correlation systems.

The aforementioned tools have been tested over different datasets — that are thoroughly documented in this document — and lead to interesting results.

Contents

List of Figures viii

List of Tables x

List of Acronyms xiii

1 Introduction 1
- 1.1 Todays' Security Threats 3
  - 1.1.1 The Role of Intrusion Detection 4
- 1.2 Original Contributions 8
  - 1.2.1 Host-based Anomaly Detection 8
  - 1.2.2 Web-based Anomaly Detection 9
  - 1.2.3 Alert Correlation 10

2 A Chapter of Examples 11
- 2.1 A Table 11
- 2.2 Code 11
- 2.3 A Sideways Table 12
- 2.4 A Figure 14
- 2.5 Bulleted List 14
- 2.6 Numbered List 14
- 2.7 A Description 14
- 2.8 An Equation 15
- 2.9 A Theorem, Proposition & Proof 15
- 2.10 Definition 16
- 2.11 A Remark 16
- 2.12 An Example 16
- 2.13 Note 16

List of Figures

1.1 Illustration taken from (?) and ©2005 IEEE. Authorized license limited to University of California. 4

2.1 : distribution of the number of other system calls among two system calls (i.e., distance between two consecutive). . 14

List of Tables

2.1 Duality between misuse- and anomaly-based intrusion detection techniques. 11
2.2 Taxonomy of the selected state of the art approaches for network-based anomaly detection. 13

List of Acronyms

DoS Denial of Service 5

HTTP HyperText Transfer Protocol 5

IDS Intrusion Detection System 6

ID Intrusion Detection 8

ISP Internet Service Provider 6

IP Internet Protocol 15

SOM Self Organizing Map 15

SQL Structured Query Language 6

TCP Trasmission Control Protocol 15

TTL Time To Live 15

URL Uniform Resource Locator 1

Colophon

This document was typeset using the typesetting system created by the Non-Roman Script Initiative and the memoir class created by Peter Wilson. The body text is set 10pt with Adobe Caslon Pro. Other fonts include , and. Most of the drawings are typeset using the packages by Till Tantau.

Introduction 1

Network connected devices such as personal computers, mobile phones, or gaming consoles are nowadays enjoying immense popularity. In parallel, the Web and the humongous amount of services it offers have certainly became the most ubiquitous tools of all the times. counts more than 250 millions active users of which 65 millions are using it on mobile devices; not to mention that more than 1 billion photos are uploaded to the site *each month* (?). And this is just one, popular website. One year ago, estimated that the approximate number of unique *Uniform Resource Locators* (URLs) is 1 trillion (?), while has stocked more than 70 million videos as of March 2008, with 112,486,327 views just on the most popular video as of January 2009 (?). And people from all over the world inundate the Web with more than 3 million tweets *per day*. Not only the Web 2.0 has became predominant; in fact, thinking that on December 1990 the Internet was made of *one* site and today it counts more than 100 million sites is just astonishing (?).

The Internet and the Web are huge (?). The relevant fact, however, is that they both became the most advanced workplace. Almost every industry connected its own network to the Internet and relies on these infrastructures for a vast majority of transactions; most of the time monetary transactions. As an example, every year looses approximately 110 millions of US Dollars in ignored ads because of the *"I'm feeling lucky"* button. The scary part is that, during their daily work activities, people typically pay poor or no attention at all to the risks that derive from exchanging any kind of information over such a complex, interconnected infrastructure. This is demonstrated by the effectiveness of social engineering (?) scams carried over the Internet or the phone (?). Recall that 76% of the phishing is related to finance. Now, compare this landscape to what the most famous security quote states.

> "The only truly secure computer is one buried in concrete, with the power turned off and the network cable cut".
> —*Anonymous*

In fact, the Internet is all but a safe place (?), with more than 1,250 *known* data breaches between 2005 and 2009 (?) and an estimate of 263,470,869 records stolen by intruders. One may wonder why the advance of research in computer security and the increased awareness of governments and public institutions are still not capable of avoiding such incidents. Besides the fact that the aforementioned numbers would be order of magnitude higher in absence of countermeasures, todays' security issues are, basically, caused by the combination of two phenomena: the high amount of software vulnerabilities and the effectiveness of todays' exploitation strategy.

software flaws — (un)surprisingly, software is affected by vulnerabilities. Incidentally, tools that have to do with the Web, namely, browsers and 3rd-party extensions, and web applications, are the most vulnerable ones. For instance, in 2008, reported around 115 security vulnerabilities for , 366 for 's (?). Office suites and e-mail clients, that are certainly the must-have-installed tool on every workstation, hold the second position (?).

massification of attacks — in parallel to the explosion of the Web 2.0, attackers and the underground economy have quickly learned that a sweep of exploits run against *every* reachable host have more chances to find a vulnerable target and, thus, is much more profitable compared to a single effort to break into a high-value, well-protected machine.

These circumstances have initiated a vicious circle that provides the attackers with a very large pool of vulnerable targets. Vulnerable client hosts are compromised to ensure virtually unlimited bandwidth and computational resources to attackers, while server side applications are violated to host malicious code used to infect client visitors. And so forth. An old fashioned attacker would have violated a single site using all the resources available, stolen data and sold it to the underground market. Instead, a modern attacker adopts a "vampire" approach and exploit client-side software vulnerabilities to take (remote) control of million hosts. In the past the diffusion of malicious code such as viruses was sustained by sharing of infected, cracked software through floppy or compact disks; nowadays, the Web offers unlimited, public storage to attackers that deploy their exploit on compromised websites.

Thus, not only the type of vulnerabilities has changed, posing virtually every interconnected device at risk. The exploitation strategy created new types of threats that take advantage of classic malicious code patterns but in a new, extensive, and tremendously effective way.

1.1 Todays' Security Threats

Every year, new threats are discovered and attacker take advantage of them until effective countermeasures are found. Then, new threats are discovered, and so forth. quantifies the amount of new malicious code threats to be 1,656,227 as of 2008 (?), 624,267 one year earlier and only 20,547 in 2002. Thus, countermeasures must advance at least with the same grow rate. In addition:

> [...] the current threat landscape — such as the increasing complexity and sophistication of attacks, the evolution of attackers and attack patterns, and malicious activities being pushed to emerging countries — show not just the benefits of, but also the need for increased cooperation among security companies, governments, academics, and other organizations and individuals to combat these changes (?).

Todays' underground economy run a very proficient market: everyone can buy credit card information for as low as \$0.06–\$30, full identities for just \$0.70–\$60 or rent a scam hosting solution for \$3–\$40 per week plus \$2-\$20 for the design (?).

The main underlying technology actually employs a classic type of software called *bot* (jargon for *robot*), which is not malicious *per sé*, but is used to remotely control a network of compromised hosts, called *botnet* (?). Remote commands can be of any type and typically include launching an attack, starting a phishing or spam campaign, or even

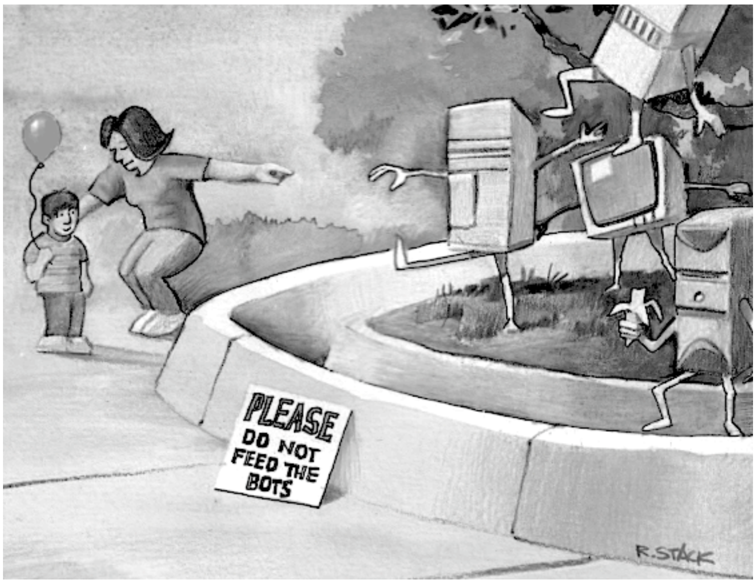

Figure 1.1: Illustration taken from (?) and ©2005 IEEE. Authorized license limited to University of California.

updating to the latest version of the bot software by downloading the binary code from a host controlled by the attackers (usually called *bot master*) (?). The exchange good has now become the botnet infrastructure itself rather than the data that can be stolen or the spam that can be sent. These are mere outputs of todays' most popular service offered for rent by the underground economy.

1.1.1 The Role of Intrusion Detection

The aforementioned, dramatic big picture may lead to think that the malicious software will eventually proliferate at every host of the Internet and no effective remediation exists. However, a more careful analysis reveals that, despite the complexity of this scenario, the problems that must be solved by a security infrastructure can be decomposed into relatively simple tasks that, surprisingly, may already have a solution. Let us look at an example.

Example 1.1.1 *This is how a sample exploitation can be structured:*

injection — *a malicious request is sent to the vulnerable web application with the goal of corrupting all the responses sent to legitimate clients from that moment on. For instance, more than one releases of the popular blog application are vulnerable to injection attacks*[1] *that allow an attacker to permanently include arbitrary content to the pages. Typically, such an arbitrary content is malicious code (e.g., JavaScript, VBSCrip, ActionScript, ActiveX) that, every time a legitimate user requests the infected page, executes on the client host.*

infection — *Assuming that the compromised site is frequently accessed — this might be the realistic case of the -powered news blog*[2] — *a significant amount of clients visit it. Due to the high popularity of vulnerable browsers and plug-ins, the client may run — that is the most popular — or an outdated release of on . This create the perfect circumstances for the malicious page to successfully execute. In the best case, it may download a virus or a generic malware from a website under control of the attacker, so infecting the machine. In the worst case, this code may also exploit specific browser vulnerabilities and execute in privileged mode.*

control & use — *The malicious code just download installs and hides itself onto the victim's computer, which has just joined a botnet. As part of it, the client host can be remotely controlled by the attackers who can, for instance, rent it, use its bandwidth and computational power along with other computers to run a distributed Denial of Service* (DoS) *attack. Also, the host can be used to automatically perform the same attacks described above against other vulnerable web applications. And so forth.*

This simple yet quite realistic example shows the various kinds of malicious activity that are generated during a typical drive-by exploitation. It also shows its requirements and assumptions that must hold to guarantee success. More precisely, we can recognize:

network activity — clearly, the whole interaction relies on a network connection over the Internet: the *HyperText Transfer Protocol* (HTTP) connections used, for instance, to download the malicious code as well as to launch the injection attack used to compromise the web server.

---

[1] http://secunia.com/advisories/23595

[2] http://wordpress.org/showcase/zdnet/

host activity — similarly to every other type of attack against an application, when the client-side code executes, the browser (or one of its extension plug-ins) is forced to behave improperly. If the malicious code executes till completion the attack succeeds and the host is infected. This happens only if the platform, operating system, and browser all match the requirements assumed by the exploit designer. For instance, the attack may succeed on and not on , although the vulnerable version of, say, is the same on both the hosts.

HTTP traffic — in order to exploit the vulnerability of the web application, the attacking client must generate malicious HTTP requests. For instance, in the case of an *Structured Query Language* (SQL) injection — that is the second most common vulnerability in a web application — instead of a regular

---

---

the web server might be forced to process a

---

that causes the page to behave improperly.

It is now clear that protection mechanisms that analyze the network traffic, the activity of the client's operating system, the web server's HTTP logs, or any combination of the three, have chances of recognizing that something malicious is happening in the network. For instance, if the *Internet Service Provider* (ISP) network adopt , a lightweight *Intrusion Detection System* (IDS) that analyzes the network traffic for known attack patterns, could block all the packets marked as suspicious. This would prevent, for instance, the SQL injection to reach the web application. A similar protection level can be achieved by using other tools such as (?). One of the problems that may arise with these classic, widely adopted solutions is if a zero day attack is used. A zero day attack or threat exploits a vulnerability that is unknown to the public, undisclosed to the software vendor, or a fix is not available; thus, protection mechanisms that merely blacklist known malicious activity immediately become ineffective. In a similar vein, if the client is protected by an anti-virus, the infection phase can be blocked. However, this countermeasure is once again successful only if the anti-virus is capable of recognizing the malicious code, which assumes that the code is known to be malicious.

Ideally, an effective and comprehensive countermeasure can be achieved if all the protection tools involved (e.g., client-side, server-side, network-side) can collaborate together. For instance, if a website is publicly reported to be malicious, a client-side protection tool should block all the content downloaded from that particular website. This is only a simple example.

Thus, countermeasures against todays' threats already exist but are subject to at least two drawbacks:

- they offer protection only against known threats. To be effective we must assume that all the hostile traffic can be enumerated, which is clearly an impossible task.

  > Why is "Enumerating Badness" a dumb idea? It's a dumb idea because sometime around 1992 the amount of Badness in the Internet began to vastly outweigh the amount of Goodness. For every harmless, legitimate, application, there are dozens or hundreds of pieces of malware, worm tests, exploits, or viral code. Examine a typical antivirus package and you'll see it knows about 75,000+ viruses that might infect your machine. Compare that to the legitimate 30 or so apps that I've installed on my machine, and you can see it's rather dumb to try to track 75,000 pieces of Badness when even a simpleton could track 30 pieces of Goodness (?).

- they lack of cooperation, which is crucial to detect global and slow attacks.

This said, we conclude that classic approaches such as dynamic and static code analysis and IDS already offer good protection but industry and research should move toward methods that require little or no knowledge. In this work, we indeed focus on the so called anomaly-based approaches, i.e., those that attempt to recognize the threats by detecting any variation from a system's normal operation, rather than looking for signs of known-to-be-malicious activity.

1.2 Original Contributions

Our main research area is *Intrusion Detection* (ID). In particular, we focus on anomaly-based approaches to detect malicious activities. Since todays' threats are complex, a single point of inspection is not effective. A more comprehensive monitoring system is more desirable to protect both the network, the applications running on a certain host, and the web applications (that are particularly exposed due to the immense popularity of the Web). Our contributions focus on the mitigation of both host-based and web-based attacks, along with two techniques to correlate alerts from hybrid sensors.

1.2.1 Host-based Anomaly Detection

Typical malicious processes can be detected by modeling the characteristics (e.g., type of arguments, sequences) of the system calls executed by the kernel, and by flagging unexpected deviations as attacks. Regarding this type of approaches, our contributions focus on hybrid models to accurately characterize the behavior of a binary application. In particular:

- we enhanced, re-engineered, and evaluated a novel tool for modeling the normal activity of the Linux 2.6 kernel. Compared to other existing solutions, our system shows better detection capabilities and good contextualization of the alerts reported.

- We engineered and evaluated an IDS to demonstrate that the combined use of (1) deterministic models to characterize a process' control flow and (2) stochastic models to capture normal features of the data flow, lead to better detection accuracy. Compared to the existing deterministic and stochastic approaches separately, our system shows better accuracy, with almost zero false positives.

- We adapted our techniques for forensics investigation. By running experiments on real-world data and attacks, we show that our system is able to detect hidden tamper evidence although sophisticated anti-forensics tools (e.g., userland process execution) have been used.

1.2.2 Web-based Anomaly Detection

Attempts of compromising a web application can be detected by modeling the characteristics (e.g., parameter values, character distributions, session content) of the HTTP messages exchanged between servers and clients during normal operation. This approach can detect virtually any attempt of tampering with HTTP messages, which is assumed to be evidence of attack. In this research field, our contributions focus on training data scarcity issues along with the problems that arise when an application changes its legit behavior. In particular:

- we contributed to the development of a system that learns the legit behavior of a web application. Such a behavior is defined by means of features extracted from 1) HTTP requests, 2) HTTP responses, 3) SQL queries to the underlying database, if any. Each feature is extracted and learned by using different models, some of which are improvements over well-known approaches and some others are original. The main contribution of this work is the *combination* of database query models with HTTP-based models. The resulting system has been validated through preliminary experiments that shown very high accuracy.

- we developed a technique to automatically detect legit changes in web applications with the goal of suppressing the large amount of false detections due to code upgrades, frequent in todays' web applications. We run experiments on real-world data to show that our simple but very effective approach accurately predict changes in web applications and can distinguish good *vs.* malicious changes (i.e., attacks).

- We designed and evaluated a machine learning technique to aggregate IDS models with the goal of ensuring good detection accuracy even in case of scarce training data available. Our approach relies on clustering techniques and nearest-neighbor search to look-up well-trained models used to replace under-trained ones that are prone to overfitting and thus false detections. Experiments on real-world data have shown that almost every false alert due to overfitting is avoided with as low as 32-64 training samples per model.

Although these techniques have been developed on top of a web-based anomaly detector, they are sufficiently generic to be easily adapted to other systems using learning approaches.

1.2.3 Alert Correlation

IDS alerts are usually post-processed to generate compact reports and eliminate redundant, meaningless, or false detections. In this research field, our contributions focus on unsupervised techniques applied to aggregate and correlate alert events with the goal of reducing the effort of the security officer. In particular:

- We developed and tested an approach that accounts for the common measurement errors (e.g., delays and uncertainties) that occur in the alert generation process. Our approach exploits fuzzy metrics both to model errors and to construct an alert aggregation criterion based on distance in time. This technique has been show to be more robust compared to classic time-distance based aggregation metrics.
- We designed and tested a prototype that models the alert generation process as a stochastic process. This setting allowed us to construct a simple, non-parametric hypothesis test that can detect whether two alert streams are correlated or not. Besides its simplicity, the advantage of our approach is to not requiring any parameter.

The aforementioned results have been published in the proceedings of international conferences and international journals.

A Chapter of Examples 2

2.1 A Table

Feature	MISUSE-BASED	ANOMALY-BASED
Modeled activity:	Malicious	Normal
Detection method:	Matching	Deviation
Threats detected:	Known	Any
False negatives:	High	Low
False positives:	Low	High
Maintenance cost:	High	Low
Attack desc.:	Accurate	Absent
System design:	Easy	Difficult

Table 2.1: Duality between misuse- and anomaly-based intrusion detection techniques. Note that, an anomaly-based IDS can detect "Any" threat, under the assumption that an attack always generates a deviation in the modeled activity.

2.2 Code

1
2
3
4
5
6
7
8
9

2.3 A Sideways Table

Approach	Time	Header	Payload	Stochastic	Determ.	Clustering
(?)		●				●
(?)		●	●	●		
(?)		●		●	●	
(?)			●			●
(?)	●		●		●	
(?)		●	●			●
(?)			●	●		
(?)		●	●			●
(?)		●	●			●
(?)			●	●		

Table 2.2: Taxonomy of the selected state of the art approaches for network-based anomaly detection.

2.4 A Figure

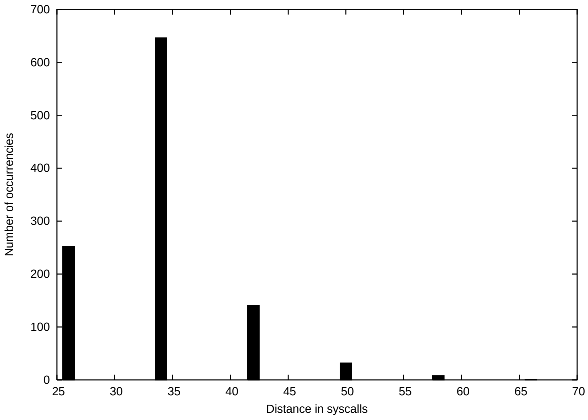

Figure 2.1: : distribution of the number of other system calls among two system calls (i.e., distance between two consecutive).

2.5 Bulleted List

- $O =$“Intrusion”, $\neg O =$“Non-intrusion”;
- $A =$“Alert reported”, $\neg A =$“No alert reported”.

2.6 Numbered List

1. $O =$“Intrusion”, $\neg O =$“Non-intrusion”;
2. $A =$“Alert reported”, $\neg A =$“No alert reported”.

2.7 A Description

Time refers to the use of *timestamp* information, extracted from network packets, to model normal packets. For example, normal packets may be modeled by their minimum and maximum interarrival time.

Header means that the *Trasmission Control Protocol* (TCP) header is decoded and the fields are modeled. For example, normal packets may be modeled by the observed ports range.

Payload refers to the use of the payload, either at *Internet Protocol* (IP) or TCP layer. For example, normal packets may be modeled by the most frequent byte in the observed payloads.

Stochastic means that stochastic techniques are exploited to create models. For example, the model of normal packets may be constructed by estimating the sample mean and variance of certain features (e.g., port number, content length).

Deterministic means that certain features are modeled following a deterministic approach. For example, normal packets may be only those containing a specified set of values for the *Time To Live* (TTL) field.

Clustering refers to the use of clustering (and subsequent classification) techniques. For instance, payload byte vectors may be compressed using a *Self Organizing Map* (SOM) where class of different packets will stimulate neighbor nodes.

2.8 An Equation

$$d_a(i,j) := \begin{cases} K_a + \alpha_a \delta_a(i,j) & \text{if the elements are different} \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

2.9 A Theorem, Proposition & Proof

Theorem 2.9.1 $a^2 + b^2 = c^2$

Proposition 2.9.2 $3 + 3 = 6$

Proof 2.9.1 *For any finite set* $\{p_1, p_2, ..., p_n\}$ *of primes, consider* $m = p_1 p_2 ... p_n + 1$. *If* m *is prime it is not in the set since* $m > p_i$ *for all* i. *If* m *is not prime it has a prime divisor* p. *If* p *is one of the* p_i *then* p *is a divisor of* $p_1 p_2 ... p_n$ *and hence is a divisor of* $(m - p_1 p_2 ... p_n) = 1$, *which is impossible; so* p *is not in the set. Hence a finite set* $\{p_1, p_2, ..., p_n\}$ *cannot be the collection of all primes.*

2.10 Definition

Definition 2.10.1 (Anomaly-based IDS) *An* anomaly-based IDS *is a type of IDS that generate alerts $\mathbb{A}$ by relying on normal activity profiles.*

2.11 A Remark

Remark 1 *Although the network stack implementation may vary from system to system (e.g., and platforms have different implementation of TCP).*

2.12 An Example

Example 2.12.1 (Misuse *vs.* Anomaly) *A misuse-based system M and an anomaly-based system A process the same log containing a full dump of the system calls invoked by the kernel of an audited machine. Log entries are in the form:*

2.13 Note

Note 2.13.1 (Inspection layer) *Although the network stack implementation may vary from system to system (e.g., and platforms have different implementation of TCP), it is important to underline that the notion of IP, TCP, HTTP* packet *is well defined in a system-agnostic way, while the notion of* operating system activity *is rather vague and by no means standardized.*