Network Surveillance Systems: Models and Approaches

by

Daniel Pestov

A Thesis submitted to the Faculty

in partial fulfillment

of the requirements for the

BACHELOR OF ARTS

Accepted

_____

Paul Shields, Thesis Advisor

_____

Michael Bergman, Major Advisor

Simon's Rock College

Great Barrington, Massachusetts

2008

Table of Contents

Introduction 1
1 Attacker's Inventory 7
1.1 IP Spoofing 7
1.2 TCP Session Hijacking 8
1.3 Denial of Service 9
1.3.1 Bandwidth Depletion 11
1.3.2 Resource Depletion 12
1.3.3 DoS Response 13
1.4 Network Probes 14
1.5 Shellcode 16
1.6 Polymorphic shellcode 18
2 Signature Detection 22
2.1 Approaches 23
2.1.1 String Matching 23
2.1.2 State Modeling 23
2.1.3 Expert System 24
2.1.4 Colored Petri Nets 25
2.1.5 Rule-Based Systems 27
3 Anomaly Detection 29
3.1 First Intrusion Detection Model 32
3.2 Sources of Audit Data 34
3.3 Protocol Anomalies 36
3.4 Statistical Anomalies 38
3.5 Approaches 38
3.5.1 Protocol State Modeling 39
3.5.2 Signal Processing of MIB Variables 42
3.5.3 Data Mining Using Clusters 43
3.5.4 Detecting SYN Floods 47
3.6 Hybrid Intrusion Detection 48
4 Response 49
5 NIDS for Distributed Networks 53
5.1 EMERALD 53
5.1.1 EMERALD Monitor Architecture 55
5.2 GrIDS 56
6 Efficiency Evaluation 59
6.1 Benchmarking 61
6.1.1 Case Study: Benchmarking Based on Data Entropy 61
7 Related Works 65
7.1 Honeypots 65
7.1.1 Honeynets 68
7.1.2 Dynamic Honeypots 71
7.2 Honeycomb 72
7.3 Identifying the Source of DoS Attacks 73
Conclusion 75
References 76

List of Figures

Figure 1: IDS Architecture 5

Figure 2: Smurf 12

Figure 3: State-based Intrusion scenario 24

Figure 4: CPA (representing partial ordering of events) 26

Figure 5: TCP State Machine 40

Figure 6: Anomaly detection using clusters 46

Figure 7: Hybrid IDS 48

Figure 8: Worm activity graph 57

Figure 9: Reducing departments into a single node 58

Figure 10: Honeynet architecture 69

Abstract:

This paper presents a technical overview of network intrusion detection, with a particular emphasis on anomaly detection systems. At the core of the discussion is a comparative analysis of two complementary paradigms of network surveillance: signature-based detection and anomaly-based detection. Signature-based detection systems analyze network traffic for patterns of known intrusive activity while anomaly detection systems look for deviations from normal system behavior. The discussion focuses on data models, approaches and response methodologies. We also introduce architectures of several commercial intrusion detection engines as well as complementary technologies and areas of research that characterize powerful yet non-traditional approaches to network security.

Introduction

Due to ever-increasing reliance on network-based electronic communication technology there is a growing demand for protecting the infrastructure and data which resides on the network. Network security must meet society's growing dependency on the Internet for e-commerce, banking, defense, healthcare, communications, energy management, and other critical applications which have become an indispensible part of daily living. Existing security safeguards on networks and hosts usually begin with authentication schemes such as access control mechanisms (e.g. passwords, firewalls, and encryption schemes). However, such measures provide only a limited layer of protection of the system since these authentication mechanisms can be either evaded or compromised. Misconfigured firewalls can be bypassed and passwords might be stolen or guessed. Attacks against secure systems and data can originate from malicious users or systems from outside or inside an organization.

According to CSI/FBI Computer Crime and Security Survey the total cybercrime losses in 2005 reached $130.1 million; the majority of these losses were due to viruses, unauthorized access to computer systems and theft of information [32]. According to CERT (Computer Emergency Response Team) [38], the total number of cataloged vulnerabilities in 2007 was 7,236, while only 171 were indexed in 1995. Vulnerabilities are weaknesses in protocols, network services, operating systems and nearly all types of network infrastructure, caused by bugs, misconfigurations and other flaws of various kinds. Attackers are malicious users who exploit vulnerabilities in order to gain unauthorized access into systems and networks, steal confidential information, disrupt operation of services and deny legitimate users access to resources. Fixing all

vulnerabilities is not a viable solution due to time, knowledge deficiency, and resource constraints. Networks are dynamic systems constantly changing their topology, adding new services, new protocols, and new systems over time. Furthermore, outdated assets cannot be easily replaced by new standards due to high interoperability and tight integration that pervades information systems. In many financial organizations entire departments exist only to provide support for long outdated devices which are needed by an organization.

To avoid the arms race between vulnerability discovery and patching local systems, security professionals and researchers have focused research on attack detection and recovery. One detection strategy is to assume attacks will occur eventually and the goal would be to design a system to become aware of such attacks. In the event of an attack, the system will alert responsible parties and/or attempt to repair the damage caused by it in some kind of organized framework.

Security systems that provide intrusion detection capabilities are called *Intrusion Detection Systems* (IDS). An *intrusion* is defined as an attempt by a user of a system to perform an action he/she was not legally allowed to perform [11]. More generally, an intrusion is any set of actions that attempts to compromise the integrity, confidentiality or availability of a potentially valuable resource, or to bypass the security mechanisms of a computer system or a network with the goal of negatively affecting the organization's interests.

Confidentiality, availability and integrity are the three core principles that motivate the development and deployment of intrusion detection systems. *Confidentiality* refers to efforts to conceal and deny access to information and resources

to unauthorized users. Accidental or intentional disclosure of sensitive information or personal data has a potential to destroy reputation of firms and businesses, undermine business transactions, even put people's lives in danger. Industrial firms, government and military agencies, health care providers take many steps to ensure that records containing personal data about their customers, employers, patients and operatives is shared only among authorized persons and organizations. Regulatory agencies pass legislations, such as Sarbanes-Oxley Act (SOX) and Health Insurance Portability and Accountability Act (HIPAA), requiring organizations to institute appropriate strategies of protecting privacy of personal and sensitive information [31]. Interception of information by an unauthorized party is called *snooping*. It may involve listening to communication, browsing through files, etc.

Integrity refers to efforts to prevent improper and unauthorized modification to information or resources. Loss of integrity also results when data is either accidentally or maliciously deleted, or when new data is created without approval by an authorizing party. Integrity ensures that information provided to public or stored for internal use is authentic, trustworthy and up-to-date. We note that sometimes a loss of integrity can be attributed to incompetence rather than malicious intent, but the end results are the same.

Availability means that information or resources can be used by those for whom it is intended. Availability is compromised when someone deliberately attempts to deny access to data or services provided by a system by either making it completely unavailable or undermining some subset of its functionality. Such attacks are called the *Denial of Service* attacks and are often very difficult to detect.

Intrusion detection systems monitor a specific target or a list of targets. Each target is a dynamic system that generates events of interest. It could be a workstation, a web server, a database, an FTP service, a routing device, an entire network, etc. *Events* are auditable changes in the state of the system. An event can represent a single action by a user or a system, or a series of actions, that when performed in sequence, produce a single observable record. Some examples of events include transmission or interception of a network packet, user running a command, database transaction, login attempt, etc. Events monitored by an intrusion detection system are called *audit events*. To detect intrusion is to find evidence that it is taking place. The incidence of intrusion is either reflected in the audit trail it generates or the subsequent state of the system.

Once audit data has been collected, intrusion detection system employs one of two possible analysis methodologies: signature-based or anomaly-based. *Signature-based detection*, also known as *misuse detection*, uses knowledge accumulated about the intrusive process and looks for specific traces it ought to leave in the target system. In other words signature-based systems define illegal behavior and each signature represents a specific attack scenario. As a result such systems achieve high degree of accuracy in detecting even the most subtle attacks. The major shortcoming of this approach is that it is limited to only known attacks.

Anomaly-based detection systems (ADS) on the other hand, assume no prior knowledge about the attacks they are expected to detect. Instead, they identify events that appear to be anomalous with respect to normal system behavior. The principal assumption confirmed by practical application is that abnormal behavior is a sign of intrusion or system malfunction. The main advantage of anomaly detection over

signature detection is the ability to identify new and unforeseen attacks. However, not every anomaly is triggered by intrusion. Therefore, anomaly-based detection systems are known for generating high numbers of false alarms.

Intrusion detection systems are also classified in terms of their monitoring scope. *Host-based intrusion detection systems* (HIDS) monitor the behavior of the individual machine on which they are deployed. Such systems typically analyze file system activity, system call invocations, application logs, etc. *Network-based intrusion detection systems* (NIDS) identify intrusions by examining network traffic and monitoring multiple hosts as well as other networking devices such as routers, switches, etc. Each individual network is a collection of computers, users and services with specific behavioral patterns that are reflected in the generated network traffic. The purpose of network-based intrusion detection is to scan network traffic for events that characterize a potential violation of security, also known as a *threat*. An actual violation of security is called an *attack*. Not all threats manifest themselves as attacks; some are false alarms, yet others are caused by malfunctioning network devices, etc.

A typical high-level architecture of intrusion detection system is illustrated in figure 1. *Detection engine* collects audit data. *Analysis engine* analyzes audit data and generates an intrusion report for every identified intrusion. *Response engine* devises and implements coutermeasures.

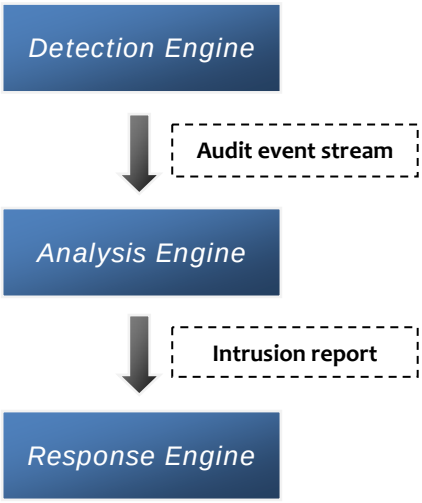

Figure 1: IDS Architecture

This thesis is an overview of network intrusion detection systems, the motivation behind them, their design and architecture. The focus will be placed on anomaly detection systems. The rest of the paper is organized as follows. Chapter 1 introduces the types of tools and techniques used by intruders today to accomplish their goals. Chapter 2 serves as a technical introduction into signature-based intrusion detection systems. Chapter 3 provides a more detailed survey of anomaly detection. In chapter 4, we discuss various response methodologies. Chapter 5 will introduce scalable network surveillance architectures. In chapter 6 we discuss how to measure the performance of intrusion detection systems. Chapter 7 will introduce various complementary solutions and related areas of research.

Chapter 1

Attacker's Inventory

Because today's computer systems are composed of many independent modules which act together to achieve the goals of an organization, the risk can take on many facets. Attacks and compromises can come from many different vectors, some of which have been around for a long time and others newly invented. Sometimes they take the form of vulnerabilities of the local software system. Other times they take advantage of misconfigurations of the network or ambiguous security settings when multiple systems interact. We now survey some popular attacks…

1.1 IP Spoofing

IP Spoofing is a technique often used by hackers that involves masking source IP address by manipulating IP header of outgoing packets. Routing devices use destination address field to route packets across the network and ignore source IP, which is only used by the destination machine when it needs to send a response. IP spoofing is useful for concealing identity, pretending to be a trusted host and carrying out many types of attacks such as denial of service, session hijacking, etc.

Few basic precautionary measures can be taken to limit the risk of IP spoofing on a network. Since most spoofing-related attacks originate from outside the target network and use IP addresses belonging to hosts inside the network, the most

reasonable solution is to deny access to any incoming packet whose source address belongs to the organization's internal IP range. This technique is called *ingress filtering* [33]. *Egress filtering* is the opposite strategy; outgoing packets are blocked if their source IP address belongs to the host outside your network. This prevents someone on the network from sending spoofed traffic to the internet. For example in the event of a machine being compromised, an organization wants to ensure their resources are not used against an outside network. Encryption and authentication mechanisms can also be used to limit most spoofing threats. Many current shortcomings are addressed in the upcoming revision of IP protocol (IPv6) once it becomes widely adopted and established.

1.2 TCP Session Hijacking

TCP Session hijacking, also known as the *man-in-the-middle* attack, is an attack that involves the interception of an already established and authenticated TCP session between two communicating parties (client and server) by an offending party.

First, the attacker intercepts one of the packets exchanged between the two machines and extracts client's IP address, port and next sequence number expected by the server. Next, they forge their own packet by spoofing client's IP and port number as well as inserting the sequence number the server expects to receive from the client on the next transmission. In order to assume client's identity, the attacker sends this packet to the server. The server will assume that they are still communicating with the client since in some cases they identify client only by IP address and port number.

There is still one problem: the client is still an active participant in this TCP session, so how can the attacker guarantee that the server only accepts their own packets and not the client's? The solution lies in the fact that TCP protocol uniquely identifies each incoming packet within the same session by its sequence number. When two packets arrive having the same sequence number, the latter is always disregarded and dropped. The attacker can desynchronize client's communication with the server by always sending his own packets ahead of client's packets with the same sequence number.

When the TCP session is successfully intercepted, neither the client nor the server is aware that there is an intermediary present. Since the server treats the client as a trusted host, the attacker who assumes client's identity can fool the server into disclosing private information or running commands on his behalf.

Another variation of this attack is to forge network identifiers so that the client thinks the malicious machine is the destination machine. For example, if a client is trying to connect to something.com, the attacker will make believe they are something.com. When the client attempts to connect to something.com, the attacker will connect to something.com on their behalf, and pass back the intended information. At the same time the attacker is free to both monitor and modify any information. This is disastrous if the client is connecting to a bank account or other sensitive data sources.

1.3 Denial of Service

Denial of service attack (*DoS*) is an attempt to compromise the effective operation or availability of a network resource or service to its intended users. There are two classes

of denial of service attacks: bandwidth depletion and resource depletion [34]. *Bandwidth depletion* attacks, also known as *flooding*, saturate the target network with a large amount of traffic, using up all available bandwidth and preventing legitimate requests from reaching their destination host. *Resource depletion* attacks trigger excessive consumption of computational resources (CPU, memory, disk space) of the target server, undermining its ability to process user requests.

The victim of a DoS attack can be any computer system which interacts with the outside. Examples include:

- Network server or client
- Internet service provider
- Low-level network device (router, switch, or proxy)
- Individual network link or an entire network
- VOIP phone
- Individual internet user or a company conducting its business using the Internet.

One low-level example is attacking necessary components of network protocol. For example the shutdown of the entire network can be achieved by simply disabling or making inaccessible a Domain Name Lookup (DNS) system. This can be accomplished by sending small DNS queries containing spoofed IP address of the target computer to each DNS server on the network. Large responses are sent to the target, resulting in link congestion and potential denial of internet connectivity. By sending many DNS queries, the network link to DNS server may become flooded with excessive traffic and consequently the DNS response time will be degraded, perhaps to the point of making

the entire network unusable. An attacker can generate traffic surges using his own host or by using other compromised machines on his behalf. Such coordinated large-scale DoS attack is called *Distributed Denial of Service* (DDoS). Each secondary system that the attacker controls and uses to wage his attack is called an *agent*.

1.3.1 Bandwidth Depletion

ICMP is a network layer protocol that facilitates the transmission of status and error messages between computers on a network. Pinging is a process of discovering live hosts on a network. It is accomplished by sending ICMP "echo request" packets to the target host and listening for ICMP "echo response" replies [9]. *ICMP flood* results when multiple ICMP pings overload a system by forcing it to redirect all its available resources to issuing echo responses, often to the point that legitimate network traffic can no longer be serviced. In order to conceal his identity (or the identity of agents in the event of DDoS), the attacker can supply spoofed IP address to all echo requests and direct all echo responses to another system.

A `smurf` is a well-known variant of ICMP flood attack with an amplified damaging effect. A `smurf` program generates ICMP echo request packet with spoofed source IP address and sends it to a broadcast address (See figure 2). A broadcast address is a special IP address that allows packets to be sent to all machines on a given subnet rather than a specific host. All the echo responses are sent back to the spoofed address (a victim host). If enough pings are generated and broadcasted this way, the entire network could be brought to a state that it can no longer respond to a legitimate traffic.

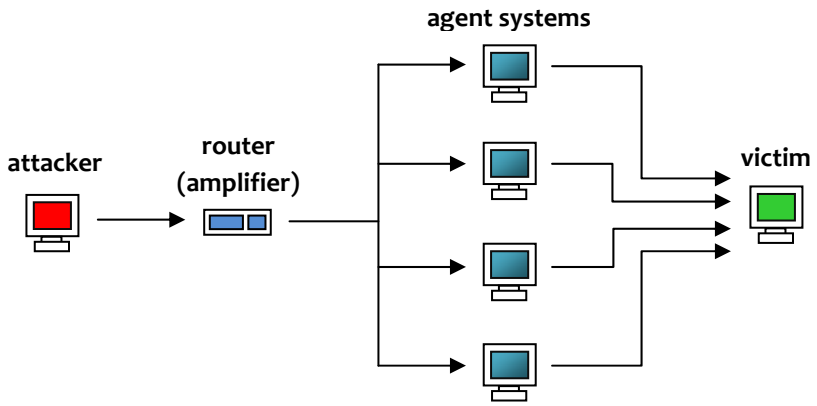

Figure 2: Smurf

1.3.2 Resource Depletion

SYN flood, also known as `Neptune`, is a type of DoS attack that exploits TCP three-way handshake [19]. The usual procedure for initiating a TCP connection with a server is by sending a SYN packet to which a server responds with a SYNACK packet. The connection is completed when the client sends a final acknowledgment (ACK packet). The problem, however, is that most servers don't wait for a final ACK; they allocate resources as soon as they receive a SYN. An attacker could exploit this fact by sending a series of SYN packets and creating many half-opened connections that consume resources. Idle TCP connections eventually expire, but if the attacker can match the rate of expiration with the rate of flooding, he could indefinitely suspend the operation of the server, effectively denying service to all other clients who wish to access its resources or use its services.

LAND attack is a type of DoS attack that involves sending a specially crafted spoofed SYN packet to a target machine where destination IP/port matches source IP/port. This causes target to reply to itself continually, resulting in a system lockup.

1.3.3 DoS Response

Once DoS attack has been detected by NIDS, the alarm is generated and network administrators are notified. Two courses of action can be taken at this point: (1) restore normal network operation by alleviating the traffic load or (2) identify the source of the attack. There are several ways to counter the effects of DoS flooding:

- *Throttling*: deploy a firewall to limit the incoming bandwidth and filter out specific packet types used in the attack (ICMP, SYN). For example, to mitigate SYN flood, a firewall can monitor the number of SYNs to a server over a specific time interval and not allow the number to exceed a configured threshold. The side effect of this technique is that legitimate traffic will also get dropped.
- *SYN Proxy*: Another protection against SYN flood is to deploy a proxy that intercepts TCP connection requests. For any half-opened connection, the proxy would automatically issue an RST (reset) packet, which will close the TCP connection and free up server's resources.
- *Load balancing*: in the event of the flood, redirect traffic to multiple machines as a way of reducing link congestion and freeing up server resources. Use some form of scheduling to assign new connection requests to different servers. New servers are added or removed according to the rate of flooding.

1.4 Network Probes

Many attacks scenarios require attackers to gain information about the target network as a preliminary to the actual attack. *Probes* or *scanners* are programs and techniques that systematically scan the host or an entire network for vulnerabilities that can be exploited. Scanning facilitates the detection of live hosts, open ports, active TCP/UDP services and native operating system. *Ping sweeping* determines if a given IP address belongs to a live host. *Port scanning* identifies listening ports on a target system and services running on them. Network services "listen" for connections at well-known ports. There are 65535 distinct and usable ports. *Fingerprinting* is a process of detecting host operating systems.

ICMP sweep is a simple ping sweep that utilizes ICMP (Internet Control Message Protocol) [1]. ICMP sweep consists of ICMP ECHO request packets sent to multiple hosts on the network, usually in a short period of time. ICMP ECHO reply packet is issued when a specified IP address belongs to a live host. A single ICMP ECHO request could also be sent to a broadcast address which multiplies the packet and sends it to every single host on a given subnet.

TCP sweep is another common ping sweep. It is accomplished by sending TCP SYN packet to common TCP ports [1]. If the server is live and the port is set to listen, the service running on it will acknowledge the connection request. If the port is not active, the server will send RESET segment in order to tear down the connection. Receiving any response is a good indication that IP address is assigned to a live host.

SYN scan is a port scanning technique accomplished by sending a SYN packet to every port on a system [1]. Reply in a form of SYN/ACK packet indicates that the port is

open, while the reception of RST/ACK packet indicates that the given port is closed. As soon as the connection is acknowledged by the server, the attacker immediately tears it down in order to avoid logging mechanisms.

Sometimes scanning is accomplished by sending unexpected or invalid packet to the target system. In such cases TCP specification requires target operating system to respond with RESET packet for ports that are closed, and send nothing if port is open. Common probe packets are FIN, SYN/ACK, XMAS (all flags set) and NULL (none of the flags are set).

`Nmap` (Network Mapper) is a famous utility that provides advanced port scanning capabilities. For each given port, it returns its state (open, closed or filtered) and service running on it (ftp, http, tcpmux, etc). Filtered port indicates a presence of a firewall, which prohibits access to certain open ports. Normally a host would respond with a SYN/ACK in response to SYN packet if the port is open and RST if the port is closed. Firewalls make it impossible to differentiate between filtered and closed ports by blocking both types of responses. There is however a walkaround called *firewalking*. The idea is to set TTL (time to live) option on a TCP/IP packet to the number of hops it takes to reach the target host. If firewall allows incoming traffic (open port), then by the time the packet reaches the target system, its TTL will reach 0 and server will send a Time Exceeded ICMP control packet back to the attacker. If the port is closed, the firewall will simply drop the datagram and the attacker will not receive any response.

Stack fingerprinting is a technique of discovering remote operating system [1]. Operating system vendors do not follow official TCP/IP specifications exactly as published in RFCs. This allows hackers to send special TCP packets to the target and

determine OS running on it by observing the responses, which will vary depending on the OS. We present two examples [10]:

- FIN probe packets sent to an open port are supposed to be ignored according to official guidelines. Some systems, however respond with a RESET packet (Windows, BSDI, CISCO).
- Operating systems can be differentiated based on which TCP options they implement. When a host responds to a query, the reply will only contain those options which are supported. By sending a TCP query with all options set, the attacker can determine operating system by observing which options were set in the reply.

So far we have discussed active fingerprinting based on probing. There is also *passive fingerprinting* based on *sniffing* [24]. Unlike probes which send malformed packets in order to illicit a specific response from a remote host, a sniffer captures outgoing packets and passively examines their headers. `p0f` is a popular passive fingerprinting tool that compares captured packets against a database of signatures. Each signature encodes values of packet header fields that are unique for a TCP/IP stack of a given OS.

1.5 Shellcode

Shellcode is a small segment of code, usually written in assembly language, which acts as a payload injected into a computer system via vulnerability in software with network interface. Once executed, it opens a command-line interface (root shell) on a target system and runs commands on behalf of the attacker. The instructions will often disable

security mechanisms, open a backdoor to allow remote access, or facilitate the transfer and installation of additional malicious code to further compromise the system.

Buffer overflow is a condition that occurs when an application receives more input that it can store inside its fixed-sized input buffer. This problem has been around for over thirty years, and still plagues computer systems and services. A common example: many programs written in C misuse library functions, such as `strcpy` and `sprintf`, and do not check the length of their arguments. Buffer overflow vulnerability results when a program accepts input string from a user and copies it into a local buffer using these calls without explicit bound checking. If the attacker provides a string larger than the allocated buffer, the adjacent memory will be overwritten. If the supplied string contains bytes that are actual native CPU instructions, the attacker can take control of the program and execute arbitrary code. Furthermore, if the program is running in privileged mode, the attacker's code inherits those privileges.

All local buffers are stored in a portion of program memory called the stack, or execution stack. The stack is a dynamic data structure that stores information associated with function calls. Each time a function is called, a new activation record is allocated on a stack that stores, among other things, all required buffers and return address – memory pointer to the next instruction that needs to be executed on function exit. In order to guarantee the execution of injected code, the attacker also needs to overwrite the return address on the stack to point to the malicious code. This also requires figuring out at what address the buffer, and thus the injected code itself, will be. To increase the chances of pointing to the correct address, attackers often pad the front of the overflow string with a sequence of NOP instructions that perform no

operation, called *NOP sled* [20]. If the return address points anywhere within NOP sled, shellcode will be successfully executed. A typical shellcode exploit consists of three parts: a set of NOP instructions, payload (injected code) and return address. It can be conceptually represented as:

[NOP SLED][PAYLOAD][RETURN ADDRESS]

1.6 Polymorphic shellcode

Since signature-based detection systems identify attack instances by matching specific patterns encoded in their knowledge base against those observed in live traffic, signature-based detection can be bypassed by generating malicious code which is functionally equivalent to the original, but syntactically altered. Attacks that have been modified for the purpose of being evaded by NIDS are called *polymorphic attacks*.

Traditionally the detection of most well-known shellcode injection attacks was carried out by simple string-based signature matching techniques. In polymorphic shellcode attacks [20], the attacker can chose an arbitrary encryption algorithm to encrypt attack code and effectively generate a semantically equivalent version of the original buffer overflow exploit. The original signature would no longer be useful. Therefore, the new exploit would no longer be detectable by signature-based intrusion detection systems, at least until the new signature itself is added to the knowledge base.

`ADMmutate` IDS evasion tool is a well-known application that systematically generates polymorphic shellcode. The tool takes as input a buffer overflow exploit, and puts it through a mutation process in order to produce thousands of functionally

equivalent versions of the same data. Each portion of shellcode exploit is disguised in a way that circumvents signature detection but preserves overall operational semantics.

- [NOP SLED] Many signature-based NIDS sensors perform pattern matching to detect NOP strings used in known exploits. Attackers, however, found ways to apply polymorphism to hide NOP sled. It turns out there are 55 single-byte instructions on Intel architecture that can serve as operationally equivalent alternatives for NOP instruction. This fact allows polymorphic engines, like ADMmutate, to randomly generate 55^n unique n-byte sequences of no-effect instructions [20].
- [RETURN ADDRESS] Since location of the stack and, consequently, the return address is unlikely to change, it would seem reasonable to generate signatures that specifically target these well-known return address values to detect code injection attacks. However, as it turns out, return address can also be disguised using polymorphic techniques. In particular, one could modulate the least significant byte of the address field and cause the control flow to jump into arbitrary position in the NOP sled [30]. The exploit would still be executed.
- [PAYLOAD] `ADMmutate` uses simple XOR encryption with a random key to disguise payload [30]. Polymorphism can also be achieved by using multiple rounds of ciphering, randomizing their order and by using different keys for each. The decoding component is also placed inside shellcode. The execution flow must now change, such that NOP sled is directly followed by decoder

module, since it must run first in order to reassemble the payload and transfer control to it.

The shellcode now has the following conceptual structure:

[NOP SLED][DECODER][ENCRYPTED PAYLOAD][RETURN ADDRESS]

- [DECODER] Since decoder is now part of the shellcode, it must itself be masked to avoid being detected by signature-based NIDS. The following techniques accomplish decoder polymorphism: multiple code paths (permuting machine instructions to perform the same operation in different ways), randomly padding code with junk NOP instructions, out-of-order decoding and randomly generated instructions [20,30].
- In addition to techniques already mentioned, it also seems reasonable to apply padding, randomize the order of individual components as they are positioned in memory by introducing additional *jmp* instructions in between or even use multiple decoders. The following are some possible shellcode layouts [20]:

  [NOP SLED][ENCRYPTED PAYLOAD][DECODER][RETURN ADDRESS]

  [NOP SLED][DECODER1][ENCRYPTED PAYLOAD][DECODER2][RETURN ADDRESS]

  [NOP SLED][ENCRYPTED PAYLOAD][PADDING][DECODER][RETURN ADDRESS]

The ability of shellcode to transform into operationally equivalent variants undermines the effectiveness of signature-based engines to identify a large set of serious network security intrusions. Given the nature of polymorphic code, there is very little prospect in generating a single comprehensive representation of a given class of self-modifying exploit. Empirical evidence in one study [20] suggests that it would require 2^{240}

signatures to represent a class of decoders alone (assuming that decoder is 30 bytes long). For the sake of comparison, the universe only has 2^{80} atoms. This means that for all practical purposes, hackers have a limitless supply of decoders at their disposal. The intractability of modeling polymorphic behavior using string-based signature schemes suggests that creating a model of normal behavior patterns is a more sensible pursuit. It would seem that anomaly detection NIDS would be better suited for the task of detecting polymorphic exploits. The demand for signature-based detection is expected to decrease as the utilization of self-modifying shellcode becomes more prevalent.

Chapter 2

Signature Detection

Signature-based detection, also known as misuse detection, is knowledge-based; it examines network traffic for patterns of well-known attacks. It usually employs an extensive updateable catalog of attack signatures as its knowledge base to recognize attacks. Once new attacks are discovered the unique pattern associated with the attack is encoded and added to the knowledge base. An ideal signature is narrow enough to represent the defining characteristics of the attack scenario and flexible enough to encapsulate all possible variations of the attack while not mislabeling good data.

Once the database is augmented with the new signature, the system becomes armed with knowledge required to detect future instances of exactly that attack. The problem arises when the original attack for which a signature exists has been slightly modified. Depending on the specificity of the original signature or the expressiveness of the underlying model, the new variation of the attack may no longer be detectable by the old signature. Polymorphic attacks, presented in chapter 1 present a serious threat to signature-detection systems since they are designed to evade specific signatures.

Unfortunately every signature-based detection system is only as good as its knowledge base. In addition the inability to detect novel attacks for which no signatures match is a concern. Time and effort are required to construct and regularly update the signature-based pattern database, so that by the time a signature is constructed, the attack might be well under way. Furthermore, as the database grows larger in size the

overall system efficiency can be affected. All these concerns non-withstanding, signature based systems are a popular deployment strategy due to the high accuracy of such systems.

2.1 Approaches

Common approaches to implemented signature-based intrusion systems will be briefly outlined in this section. They include string matching, state modeling, expert systems, colored Petri nets, and rule-based systems.

2.1.1 String Matching

String matching is the most straightforward type of signature detection [2]. Each intrusion scenario is represented in some form as an ASCII string or some binary pattern. The system scans some data (e.g. incoming traffic (usually packet payloads)) and looks for specific patterns. If a pattern is matched, the intrusion is considered to have taken place and some action is triggered.

2.1.2 State Modeling

State modeling systems encode each intrusion scenario as an ordered list of states and transitions [2,11]. Each unique state has the following conceptual structure: [attribute_1, value_1] [attribute_2, value_2] … [attribute_n, value_n]

Each attribute or feature is a system parameter of interest monitored over time by the IDS. Actions (or events) that alter the values of these attributes initiate the

transition of the system from its current state to the next. Each intrusion scenario is represented as a state machine (see figure 3).

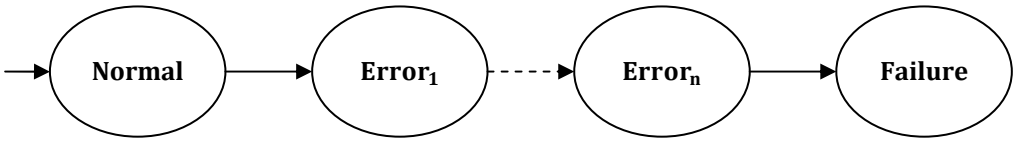

Figure 3: State-based Intrusion scenario

Each node represents a state of the system. Arrows define transitions between states. The normal state is a state of the system before an intrusion takes place. Failure state represents the completion of the intrusion. Each intermediate error state is also a part of the intrusion scenario described by the state machine.

2.1.3 Expert System

Expert system uses a set of `if-then` implication rules to describe known intrusion scenarios [13,2,4,11]. Accumulated data from audit events is converted into representational form and used as facts. Knowledge base of such system consists of fact base and rule base. At the heart of the expert system is inference engine which draws conclusions from facts using rules, usually using techniques like forward chaining. Each rule has a form [antecedent][consequent] where *antecedent* is a list of patterns that need to be matched with facts and consequent is a list of actions that will be executed if the matching is successful.

When some set of facts from a 'fact base' matches all patterns in antecedent, a binding is created for each variable-value pair where values are taken from facts that

have been matched. Among all such rule instantiations, the best one is picked using a process called conflict resolution. The bindings are then applied to consequent of that rule and the rule fires. The firing will execute various actions such as generating security alerts, terminating user session, adding new facts to the knowledge base or removing existing facts. The process keeps going until no more rules can fire.

2.1.4 Colored Petri Nets

Each attack scenario is encoded as an instantiation of Colored Petri Automaton (CPA) [13]. CPAs are a lot more expressive than regular state machines: they provide conditional matching and partial ordering of events. CPA consists of states, transitions, directed arcs that connect states with transitions, and tokens. CPA can have multiple start states and one unique final state. CPA is initialized with one token placed in each start state. The attack is considered to have take place when all tokens reach final state.

Transitions can optionally be assigned boolean expressions called *guards*. Guards permit assignments to CPA variables, evaluation of conditional operators and calling of arbitrary functions. Conceptually tokens are "colored" and each color corresponds to a unique set of patterns. Each event is "tagged" with associated parameters or facts. Patterns are unified against facts in order to generate variable bindings. These bindings are then passed to guards for evaluation.

Each transition is allowed to have multiple input states as well as multiple output states. A transition is enabled when each of its input states contains a token.

Four conditions must be satisfied in order to fire a transition:

1. It must be enabled.
2. Specified transition event must fire.
3. Variable bindings must successfully unify.
4. Guards must be evaluated to `true`.

When transition fires, all its tokens, along with the information they contain, are merged into one and the generated token is placed in the output state. If there is more than one output state, the token is duplicated and a copy is placed in each output state.

Example CPA is illustrated in figure 4. There are 7 states and 5 transitions. Each transition has an associated event label. Black transitions are enabled since their input states contain tokens. Transition of event e_5 will be enabled when there is a token in both states s_3 and s_6. Arrowed boxes symbolize guards. This CPA represents partial ordering of events, such that event e_1 necessarily precedes event e_2 and event e_3 necessarily precedes event e_4. Event e_5 is always the last in the sequence. This CPA also maintains a list of global variables. Guards can use these variables during evaluation and periodically assign new values to them.

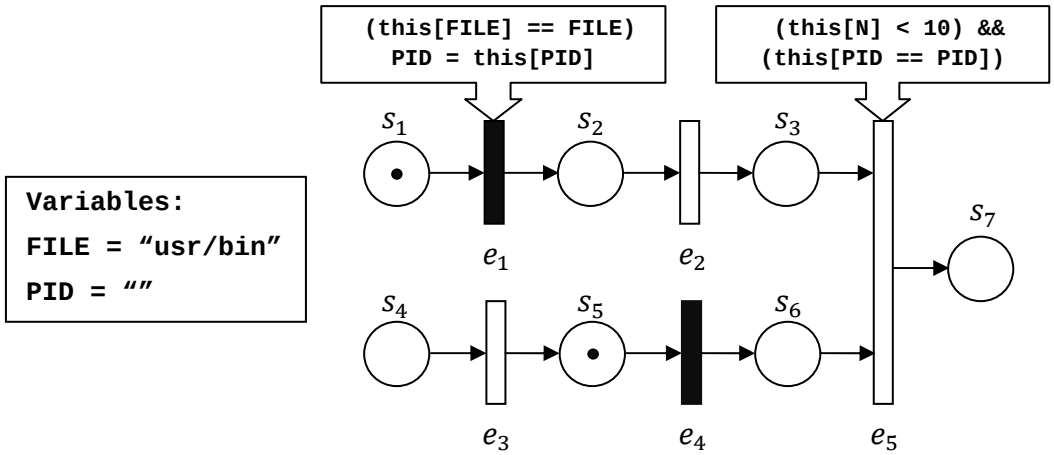

Figure 4: CPA (representing partial ordering of events)

2.1.5 Rule-Based Systems

Rule-based systems lack the power of expert systems but achieve gains in simplicity of modeling intrusions [2]. Each intrusion scenario is represented as a single rule of the form $(cond_1 \wedge cond_2 \wedge \ldots \wedge cond_n) \rightarrow actions$. Intrusion is considered to have taken place when each condition is evaluated to `true`. When a rule fires in response to intrusion, all actions are executed (logging of intrusion, alerting security staff, implementing recovery mechanism, etc).

`Snort` is a free, cross-platform, lightweight, rule-based intrusion detection system [28]. It is considered to be a cost-effective alternative to expensive commercial NIDS systems. `Snort` is implemented as a packet sniffer that monitors network interface in real time and for each intercepted packet performs content pattern matching. Each `Snort` rule is a list of packet tests and action which gets executed when all tests are satisfied. Action will either write the packet into a log or generate an alert.

`Snort` architecture consists of packet decoder, detection engine and logging/alerting subsystem [28]. Packet decoder takes packets from different networks interfaces and forwards them to the detection engine, where intrusions are detected. Detection engine matches each packet against all rules until one fires. `Snort` rules allow matching of packet payloads and individual header fields of all popular network (IP, ICMP), transport (TCP, UDP) and application-layer (DNS, FTP, SMTP, etc) protocols. In the absence of any intrusion, the packet is dropped. The logging/alerting subsystem is responsible for writing log files and generating event notifications using user-specified methods.

Snort is a simple, yet powerful intrusion detection system capable of detecting a verity of intrusions, including stealth port scans, buffer overflows, CGI attacks, and more. It is easily deployable on almost any node on the network and requires minimal administrative maintenance [28].

Chapter 3

Anomaly Detection

Anomaly detection systems (ADS) are *behavior-based*: they build a reference model of normal system behavior and intrusions are identified by detecting deviations from that model. Anomaly detection systems operate under a single assumption that if something is abnormal, it is suspicious. For example, an intruder and legitimate user of a system are likely to exhibit statistically distinct behavioral patterns.

Anomaly detection has two phases: training phase and detection phase. In the *training phase* the behavior of the system is observed in the absence of any intrusions, and profile of normal behavior is created. Unlike signature-based detection that analyses individual network events, anomaly detection is also interested in learning the dynamic statistical properties of network traffic, both on the global scale and for individual network nodes. A profile specifies how a network is supposed to behave in the absence of any attacks. It includes all learned statistical properties (dynamic knowledge) and protocol specifications (static knowledge) [6]. In *detection phase*, a profile is compared against current behavior of the system and any substantial deviations, or anomalies, are treated as indicators of a potential attack on the system.

Anomaly detection offers several advantages over signature based intrusion detection systems.

- Anomaly detection systems have the ability to detect unknown and "zero day" attacks. Intrusive activity triggers an alarm not because the system

recognizes specific attack signature, but because the deviation from normal activity is detected.

- Anomaly detection offers flexibility. Unlike signature-based detection systems that usually require a separate signature for every unique attack instantiation, a single anomaly can represent an entire class of attacks.
- Not every anomaly is triggered by malicious activities; some are caused by malfunctioning network devices which can also be detected by anomaly detection systems. Some examples of network breakdowns include network overloads, file server failures, congestions, broadcast storms, etc. *Broadcast storms* are conditions in which a packet is broadcasted to all hosts on a network, and each packet prompts a receiving host to respond by broadcasting its own packet that in turn prompt further responses, and so on. This snowball effect can have a serious negative impact on network performance. *Network congestions* usually occur due to link or node failures which consequently result in all packets being rerouted on a different link, causing excessive traffic load and substantially reducing system throughput.

Anomaly detection can also sometime suffer from serious drawbacks:

- Before anomaly detection system can be used to detect intrusive activity, it must be trained. Creating a profile of normal network behavior is a challenging task. Such profile must be abstracted to include only those features that are necessary to detect the types of attacks that the network is vulnerable to. Such information is not always available in advance and is rarely complete. Extraction of relevant informative from training data turns

out to be a difficult task because the typical behavior of any computer network is chaotic and correlations between related events are often difficult to discern. Deficient profile leads to poor performance.

- Profile must be constantly updated since network behavior is dynamic; new services are added, new systems are constantly introduced, users change their habits, adapt their behavior and are assigned new tasks. ADS system needs to account for this inherent instability by adaptively recalibrating its normal data model to reflect the new environmental conditions. Maintenance of profiles can be extremely time-consuming.
- Anomaly detection systems have poor accuracy of detection characterized by high rate of *false positives* (tendency to generate an alarm in response to activity which is legitimate, yet abnormal) and *false negatives* (failure to detect an intrusive activity which deviates only slightly from normal behavior). Valuable time is often wasted, responding to false alarms.
- In many practical settings it is impossible to guarantee that the trained data is completely attack-free. Patient attackers often purposefully train anomaly detection to gradually accept malicious behavior as normal.
- When anomaly detection system alarms network staff about a possible intrusion, it is often difficult to determine which specific event triggered the alarm. This uncertainty results in delayed response.

3.1 First Intrusion Detection Model

The first real-time anomaly-based intrusion detection model was proposed by Dorothy Denning in 1986 [5]. The model was based on the hypothesis that security intrusions affected the state of the system by introducing anomalous activity patterns. Therefore one could detect intrusions by observing and analyzing anomalies. The motivating factors for developing real-time intrusion detection model were [5]:

1. Almost every system suffers from existing security flaws which can be exploited for malicious purposes, such as gaining access to sensitive data, undermining the normal functionality of services, monopolizing resources, etc.
2. Systems with known flaws cannot be easily replaced without sacrificing some subset of their functionality or for economic reasons.
3. Developing new systems which are 100% secure is extremely difficult.
4. Even most secure systems are still vulnerable to privilege abuses by authorized users.

The proposed model is made up of the following abstract components:

- *Subjects*: initiators of actions (users, processes, systems).
- *Objects*: system resources (files, network devices, programs, database records).
- *Audit records*: actions performed by subjects on objects (file I/O, login attempts, database record retrieval).
- *Profiles* – model of normal system activity, involving the behavior of a set of subjects with respect to a set of objects. Profiles contain a set of metrics and corresponding statistical models. Metrics are random variables representing

quantitative measures sampled over a period of time. Each metric is a set of sample points that represent the value of a random variable at a particular time. These sample points are fed into the statistical model in order to determine if a new sample point raises an anomaly condition. Metrics could be event counters, interval timers between related events, or quantities specifying consumption of a particular resource.

Given a random variable x sampled n times to create sample points $x_1, x_2, \dots, x_n$, statistical model determines whether a new sample point x_{n+1} is anomalous with respect to $x_1, x_2, \dots, x_n$. There are five proposed statistical approaches that can be adopted by intrusion detection system in order satisfy this goal:

1. *Operational model* – a fixed threshold that specifies a range of values considered normal is generated manually using past experience (sample points $x_1, x_2, \dots, x_n$ are often considered). x_{n+1} is deemed abnormal if its value falls outside normal range (number of successive login failures exceeds a certain threshold).
2. *Confidence interval model* – mean μ and standard deviation σ of sample points $x_1, x_2, \dots, x_n$ is calculated in order to obtain a confidence interval $\mu \pm d \times \sigma$. x_{n+1} is deemed abnormal if its value falls outside this interval. This model is more flexible than operational model because its abnormality interval is subject-dependent.
3. *Multivariate model* – similar to confidence interval model except it correlates two or more related metrics.

4. *Markov process model* – represents each distinct type of event as a state and uses state transition matrix to characterize frequencies of transitions between any two states. An event is considered anomalous if the probability defined by previous state and transition relation is too low. Markov process model is used to detect irregularities in event sequences.
5. *Time series model* – combines interval timer and event counter. For a set of sample points $x_1, x_2, \dots, x_n$ the model takes into account their values, order and arrival times. x_{n+1} is flagged as abnormal if its probability of occurring at the time it is measured is too low. Time series model is useful for detecting gradual but substantial variations in behavior measured over time. However, it is more computationally expensive than previous models.

- *Anomaly records* – records created in response to an anomaly raised with respect to a given profile.
- *Activity rules* – conditional rules specifying actions that need to be performed when an audit record or anomaly record is generated or when a time period ends.

3.2 Sources of Audit Data

There are several approaches used to perform anomaly detection and the choice of approach depends on the nature of network data available for analysis [27]. There are

various sources that provide network performance information that can be used to detect anomalous network events [6,27]:

- Network probing tools such as *ping* or *traceroute* collect network performance measures such as end-to-end delays and packet loss levels. Such tools provide accurate data on the current state of the network.
- Routing information provides network topology and link utilization levels.
- MIB variables obtained from Simple Network Management Protocol (SNMP) are counter variables that measure traffic information at the individual network device. Provided information varies depending on where the device is located on the protocol hierarchy.
- Network and transport layer packet headers and payloads.
- Network traffic (tcpdump, NetFlow)
- Traffic probes that capture and analyze network packets. Ntop is a UNIX-based traffic probe that provides a set of counter variables that monitor various network activities, such as total traffic for specific protocols (volume and number of transmitted packets), TCP session history (duration, transmitted data, ratio of fragmented packets), running TCP/UDP services and installed operating systems, overall bandwidth consumption, traffic distribution (local vs. remote), protocol distribution (UDP vs. TCP), packet distribution (in terms of size, IP vs. non-IP), etc [6].

3.3 Protocol Anomalies

Protocol anomalies refer to all exceptions related to protocol format and behavior with respect to typical practical application. Network protocol is a set of rules governing the transmission of data between computers, applications, networks and individual communication devices. Most protocol specifications are published in RFCs and similar documents. Protocols typically monitored include transport layer protocols (TCP, UDP), network layer protocols (IP, ICMP), and application layer protocols (HTTP, FTP). It is important to note that protocols are rarely implemented in practice according to their official specifications. A model of normal usage needs to account for this fact by superposing official and practical standards of usage. Individual packet headers are examined to determine if they obey official or practical guidelines.

The order in which the packets are received also matters. Not all packets carry actual data; some packets are responsible for establishing new connections (SYN), tearing down existing connections (FIN, RST), acknowledging the receipt of data (ACK), etc. In addition to specifying legal format for each packet, protocols define conditions under which it is permissible to send packets of various types. For example, consider a typical TCP session. Any TCP connection is established by a procedure called a *three-way handshake*, where a client sends a special control packet, called SYN packet, to the server application running on a known port, in response to which the server sends SYNACK packet which acts as an acknowledgment of connection request. Finally the client sends its own acknowledgment back to the server. Following connection setup phase is data transfer phase during which client and server exchange application data.

TCP session is terminated by a *four-way handshake* during which FIN packets and their respective acknowledgments are transmitted.

The following are examples of protocol anomalies [6,18]:

- IP packets with spoofed source address
- IP packets where the source and destination address are set to address the same device (LAND attack)
- Out-of-sequence TCP packets
- Unusually large packets (ICMP Ping of Death)
- TCP packet that has unexpected or prohibited combination of flags (packet where SYN flag is set but belonging to a session already in progress)
- Fragmented IP packets when fragmentation is not required
- Invalid, overlapping or missing IP fragments
- Illegal packet flows (incomplete TCP three-way handshake that creates a half-open connection. Could be a sign of a SYN scanning or SYN flooding)
- Valid packet sent by the unexpected network agent (ICMP "redirect" packet sent by a host that isn't a router)
- Sending non-HTTP data to port 80
- Running a service on a non-standard port (HTTP packets arriving on port 53)
- Corrupt checksums

3.4 Statistical Anomalies

Statistical anomalies are detected by observing an aggregate behavior of network traffic (both globally and at individual network devices) over a period of time. In the absence of an attack, there is a stable balance among different types of outgoing and incoming packets: TCP FIN, TCP SYN, TCP data, ICMP echo request/reply, etc. Certain attacks, such as DoS attacks, will affect traffic patterns recorded during training phase and will result in statistical anomalies. Traffic patterns are typically monitored by network management protocols and traffic probes.

Some examples of statistical anomalies include [6,18,27]:

- Unusually high volume of UDP traffic relative to TCP traffic
- Burst in the frequency of timeouts (connections expiring due to inactivity)
- Unusually high volume of SYN packets relative to other types of traffic (SYN flood)
- Excessive traffic to the mail server (Possible DoS attack)
- Rise in the number of connection attempts made to a particular port
- Unusually high ICMP echo/reply ratio (sign of network probing)

3.5 Approaches

At this point we present several case studies that delineate various anomaly detection approaches used in practice.

3.5.1 Protocol State Modeling

Most known protocols can be conceptually represented as finite state machines. More formally, protocols are modeled using extended finite state automata (EFSA) [18], which differ from traditional finite state automata in two respects: (1) events of EFSA may have arguments, and (2) it can have a finite set of state variables. Each EFSA has a list of control states $\{s_1, \dots, s_n\}$, start state s, final state f, and a list of state variables $\{v_1, \dots, v_n\}$. Transitions are directed arcs, connecting states.

TCP protocol is the most widely used connection-oriented transport protocol. A separate protocol state machine is maintained for each active TCP connection. States represent various connection stages while transitions represent the event of receiving a particular type of packet or timeout. For example, when client and server complete their three-way handshake, the TCP connection enters the ESTABLISHED state. If the server receives a FIN packet requesting connection to be terminated, its TCP protocol machine will enter FIN_WAIT_1 state. For a complete description of states and transitions of TCP protocol, see figure 5.

A *transition relation* has a form $e(x_1, \dots, x_n)|cond \rightarrow [actions, state]$ where e is an event, variables $x_1, \dots, x_n$ are arguments of that event, $cond$ is a boolean expression that involves state variables, event arguments and current control state. $actions$ and $state$ are a list of actions that will be executed and a state the EFSA will enter, respectively, when event e occurs and $cond$ evaluates to *true*. Possible actions include assignments to state variables and invocations of external functions.

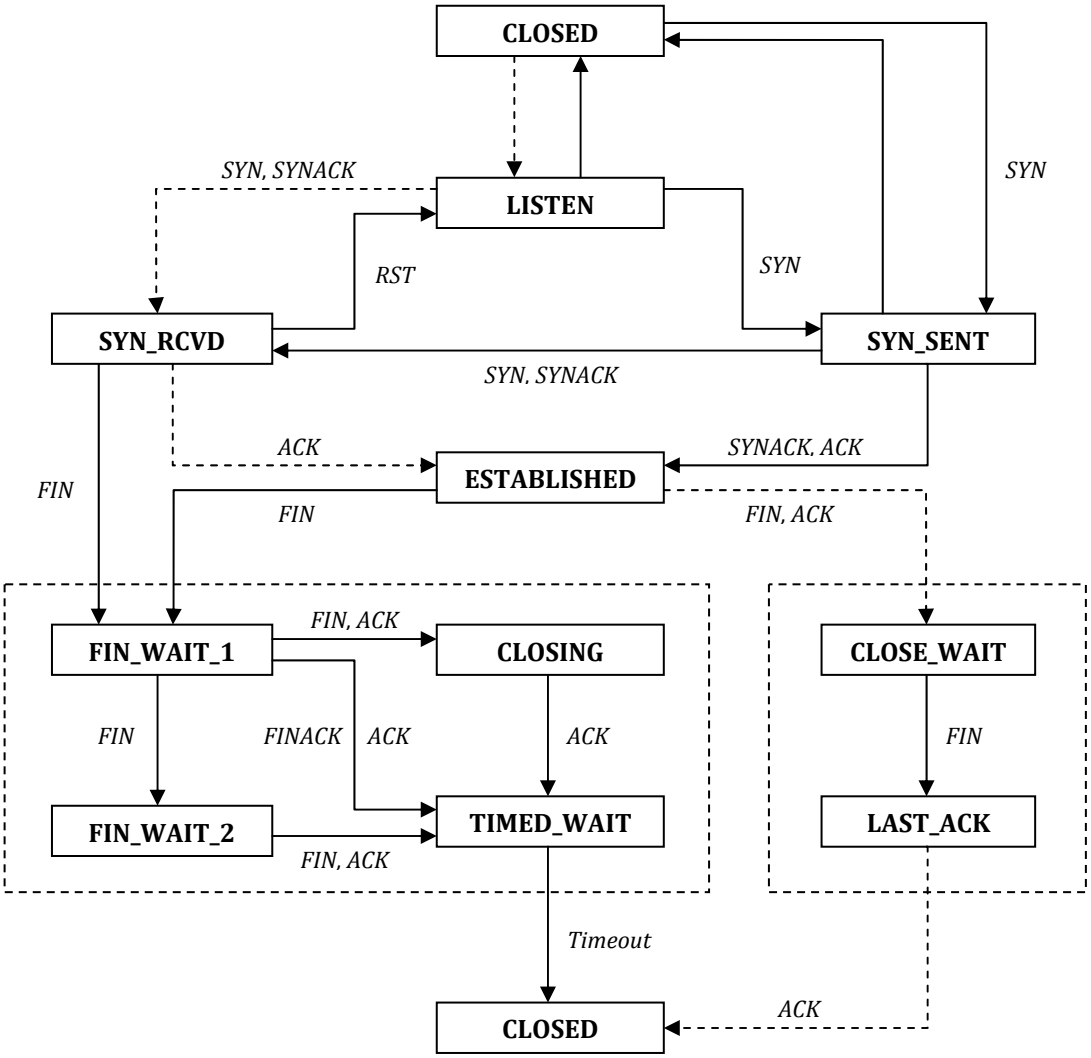

Figure 5: TCP State Machine

When monitoring behavior of a certain protocol, multiple instances of state machines are created, one for each active connection. Connection is defined by the source IP/port and destination IP/port. When a packet arrives, it is forwarded to all existing state machines. However, only the state machine whose connection matches the source of the packet will fire the appropriate transitions. If a packet initiates a new

connection request, a new state machine is allocated to keep track of the new connection. A trace is defined as a sequence of states visited by a given EFSA during its lifetime along with corresponding state variable values.

The following statistics can be obtained from traces of IP state machine [18]:

- Frequency with which each transition is taken
- Most commonly encountered value of a particular state variable at a particular state
- Distribution of values of state variables

Transitions taken by a state machine of a given trace represent two types of important events [18]: (1) the reception of unexpected packet and (2) timeout event which means that expected packet was not received. Both events suggest a possible network failure or attack. Two properties related to individual transitions are identified [18]: (1) whether a given transition is taken by a trace and (2) the value of a given state variable or a packet field when a transition is taken. This information can be represented as average values, but due to chaotic nature of network behavior, capturing distributions is more desirable as it provides a more accurate measure of network activity. Type (1) properties are captured as frequency distributions, whereas type (2) properties are captured as distributions of values of state variables. If values are categorical (IP address), as oppose to scalar (packet field size), they are represented as discrete counters. Distributions are measured on multiple timescales that range from milliseconds to thousands of seconds. This provides a balance between fast detection of rapidly progressive attacks and delayed but more accurate detection of slower attacks.

Certain statistics are specifically tailored to detect well-known attacks. For example, the number of unique IP addresses for which the packets were received in the last *t* seconds, where *t* is a small time frame, and the frequency of *timeout* transitions in PKT_RCVD (packet received) state of IP state machine, are both useful statistics for detecting ping sweeps [18].

Protocol-based anomaly detection systems generally have high detection rates of both known and unknown attacks and low false positive rates. One of the main benefits of such systems is simplified feature selection. Most attacks can be detected by simply observing the distribution of frequencies with which each transition is taken in a state machine.

3.5.2 Signal Processing of MIB Variables

Simple Network Management Protocol (SNMP) is implemented as an application layer protocol that runs over UDP and provides facilities for exchanging management information between networking devices. SNMP allows network administrators to monitor network health, as well as detect and resolve network performance issues. An SNMP-managed network consists of three vital components: network devices that need to be monitored, SNMP agents, and SNMP manager. Managed devices are individual network nodes such as routers, switches, hubs, etc. Agents are software applications that reside in a managed device and collect network management information which they then communicate to SNMP manager using SNMP protocol.

Every network device stores a set of MIB (Management Information Base) variables that are specific to its functionality and implemented as counters. Network

devices are classified in terms of how far up the protocol stack they operate. For example, routers are network-layer devices while bridges are link-layer devices.

Statistical analysis of MIB variables allows network administrators to detect many types of network-performance anomalies in an effort to anticipate and effectively prevent network-wide failures. In one study [27], signal processing method was used to detect network anomalies by modeling correlated abrupt changes in time series generated from three MIB variables chosen from IP-layer group: ipIR (the total number of datagrams received from all the interfaces of the router), ipIDe (the number of datagrams forwarded to the higher layers) and ipOR (the number of datagrams received from the higher levels). Four types of network anomalies were detected using this approach: file server failures, protocol implementation errors, network access problems and runaway processes.

3.5.3 Data Mining Using Clusters

Under normal operation, network adapter only processes packets addressed to its unique MAC address. When an adapter is switched to *promiscuous mode*, it intercepts all packets passing through its network and forwards them to the upper levels of protocol stack. `tcpdump` is a powerful packet sniffer that logs the headers of all packets that arrive at the network interface. Vast volume of training data can be collected this way and used to generate a profile of normal network behavior. However, extracting relevant information from a large amount of multidimensional raw data is a difficult and often computationally expensive task.

Data mining refers to methods and algorithms used to analyze data in order to learn about its characteristic properties. In other words, data mining is a process of extracting knowledge from data. In the context of network anomaly detection, data mining can serve two important functions: traffic classification and outlier detection. *Traffic classification* is a process of defining patterns that are typical for a particular type of network traffic, for example normal HTTP traffic, traffic observed during ICMP flood or traffic generated by SYN probing. *Outlier detection* refers to identifying singular data objects that do not belong to any existing traffic profile and therefore treated as anomalies. In this section we examine a data mining methodology called clustering.

Clustering refers to methods and algorithms that partition a set of data points into a finite set of clusters. A *cluster* is an aggregate of data objects, which are assumed to be similar to one another within the same cluster, and dissimilar to data objects of other clusters. A good clustering scheme will produce clusters with high intra-class similarity and low inter-class similarity. There are many classes of clustering algorithms described in literature: partitioning algorithms, hierarchical algorithms, density-based algorithms and grid-based methods [14]. For example, *density-based clustering* algorithms assume that clusters are regions of high density in the data space surrounded by regions of low density. For each data point, the algorithm computes the density of its neighborhood within a certain radius. All neighborhoods of high density (regions where number of data points exceeds some minimal threshold) are then aggregated together to form clusters [14]. One common technique is to partition the data space into a grid of cells, compute the density of individual cells and then merge

cells together to form clusters. This approach reduced processing time allowing the algorithm to scale to larger sets of data.

In one study [35], *K-means* partitional clustering algorithm was used to detect network anomalies by classifying flow records produced by Cisco NetFlow protocol. A *flow* refers to a unidirectional stream of IP packets identified by their source IP/port, destination IP/port and transport protocol (TCP, UDP). Each flow record also includes associated statistical properties such as total number of packets and bytes transmitted at specific time intervals.

In order to produce training data for K-means clustering algorithm, flow records need to be preprocessed and transformed. First, flow records are grouped into service classes according to utilized protocol and port numbers reserved for commonly used services. For example, web server runs on port 80 over TCP (Web/HTTP service class) while DNS uses UDP on port 53 (DNS service class). The reason for this classification is that characteristic properties of normal traffic vary across services. K-means algorithm is thus applied separately for each service class. Next, flow records belonging to individual classes are aggregated and transformed into datasets for equally spaced time intervals. The following features are defined for each dataset:

1. Total number of packets sent to or from a given port in the considered time interval
2. Total number of bytes sent to or from a given port in the considered time interval
3. Number of unique source-destination pairs observed in the considered time interval.

(1) and (2) facilitate the detection of traffic-volume anomalies (Denial of Service) while (3) helps detect anomalies associated with network probing (ICMP ping, SYN port scan) and distributed attacks.

K-means is an iterative clustering algorithm that partitions data objects into K disjoint clusters within their feature space. The algorithm follows these steps:

1. Define K arbitrarily chosen centroids (mean points).
2. Generate a set of clusters by assigning all data points to the nearest centroid.
3. Recalibrate the position of each centroid by moving it to the center of its respective cluster.
4. Repeat steps 2 and 3 until centroids converge, forming final clusters.

Since anomalous traffic and normal traffic are assumed to be characteristically different, K is chosen to be 2, meaning K-means algorithm outputs at most two centroids, one for each type of traffic. Once clusters are generated and manually labeled as either normal or anomalous, network anomalies can be detected using distance-based classification and outlier detection. A data object is classified as an anomaly if it is closer to the anomalous centroid than to the normal centroid, or if its distance to the normal centroid is larger than the predefined threshold $d_{\max}$, in which case it is treated as an outlier. In figure 6 point A is treated as anomaly because it is

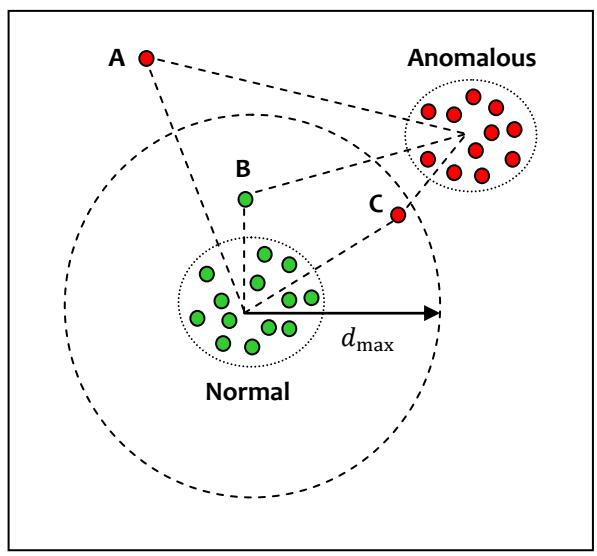

Figure 6: Anomaly detection using clusters

outside $d_{\max}$ and point C is treated as anomaly because it is closer to anomalous centroid.

3.5.4 Detecting SYN Floods

According to CSI/FBI Cybercrime Survey Report of 2003, DoS were responsible for a loss of $65 million. TCP SYN flood is the most prevalent type of DoS attack [19]. Therefore it is crucial to have effective means of detecting and dealing with SYN floods as early as possible. Adaptive threshold algorithm is a simple algorithm that facilitates the detection of SYN flood attacks by testing whether the number of SYN packets present in network traffic over a given time interval exceeds a certain threshold. The value of a threshold is adaptively modified to account for recent behavior defined by exponentially weighted moving average. Adaptive threshold algorithm performs very well for high intensity attacks, yielding high accuracy, low false alarm rate and low detection delay [19].

For low intensity attacks however, the algorithm's performance is significantly deteriorated due to its limited ability to maintain past history of threshold violations, producing high false alarm rate. Move advanced algorithms exist that perform equally well for detection of low and high intensity attacks, such as CUSUM (cumulative sum) algorithm based on hypothesis testing [19].

3.6 Hybrid Intrusion Detection

It is important to note that a reliable and robust intrusion detection system should combine both anomaly and signature-based detection. System that enjoys the complementary benefits of both types of detection methodologies will be characterized by high accuracy, ability to detect novel attacks and reduced rate of false alarms. Intrusion detection system that integrates anomaly detection and signature detection is called *hybrid detection system* [7].

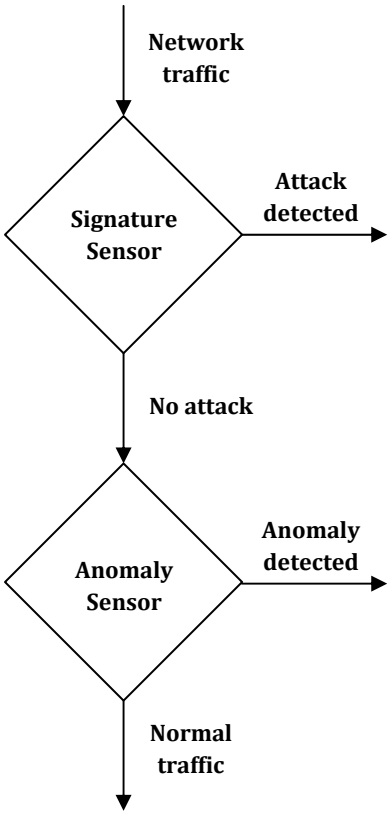

Figure 7: Hybrid IDS

Such approach divides detection into two stages. In the first stage, network traffic passes through signature sensor. Alarm is generated in the presence of an attack (assuming the attack is known in advance). Otherwise, the traffic is considered clean at least as far as signature detection is concerned. In the second stage, the same traffic is fed into anomaly sensor. If the anomaly is detected, the alarm is generated. See figure 7.

Hybrid intrusion detection is in principal a much more effective solution to network surveillance. However, different intrusion detection technologies examine network traffic in very specific ways and are configured to operate in restricted highly tuned network setups. The major challenge underlying the practice of building an operational hybrid system is getting its component systems to effectively interoperate.

Chapter 4

Response

Detecting intrusions is only the first step towards securing the network. Attack detection by itself is of limited use if no measures are implemented to issue a response or initiate some form of recovery mechanism from malicious activity. Intrusion Response Systems (IRS) obtain misuse or anomaly reports from the IDS systems in order to decide how to effectively thwart the attack and ensure the safety of computing assets. Historically, intrusion response has received a lot less attention than intrusion detection because of the need to involve a live operator in the decision loop. But as attacks proliferate in sophistication, speed, and incidence, it becomes more and more costly to neglect this area of network infrastructure.

There are two general categories of response: active and passive [17]. *Passive response systems* make no attempt to eliminate the attacks or minimize damages caused by them. Their only job is to log the intrusion or forward a note to the network management staff (e.g. via email, pager, cell phones) who would then have to decide how to respond. Intrusion reports might contain information such as attack target id, time of attack, severity level, information on specific packets used to gain unauthorized access, IP address of the attacker, anomaly statistics, etc [25]. Most intrusion detection systems have historically implemented a passive response mechanism. This can be attributed to the high false positive rates of early IDS systems and the bias towards including a human operator in the decision loop.

Active response systems, on the other hand, attempt to counter an attack by taking some form of evasive or corrective action. Active response systems are further classified in terms of their level of automation [25]. *Manual response systems* generate a list of possible countermeasures but the final decision rests with the network management staff. The main advantage of requiring human intervention is that many systems have been known to generate a high rate of false alarms. It would be counter-productive and often detrimental to the effective operation of the network if system took potentially aggressive action in response to every detected anomaly, such as closing ports or terminating live connections. Should attackers themselves learn this, they will gain the ability to trigger a denial of service to such systems by merely attempting an intrusion. However, certain networks suffer from high incidence of malicious activity and requiring human intervention on every attack might be too costly. Network administrators are known to be rather busy and delayed responses provide attackers with a sufficient window of opportunity to succeed in their attack [17]. Studies show that if the time gap between detection and response is 10 hours, skilled human attackers have an 80% success probability [36]. On the other hand, automated exploits often require mere minutes or even seconds to bring about irreparable damage [37]. In addition the type of resource being protected is a factor in the decision. Protecting an e-commerce website is different than protecting a nuclear weapons network.

Autonomous response systems automatically decide on and execute an appropriate response policy. Examples of common response actions are: modifying firewall rules and access control lists (ACL), blocking ports and IP addresses,

terminating TCP connections, tracing the source of the attack, restarting the target, diverging suspicious TCP connection from the target system to a "trap" system [25,26], etc. See section 7.1 for additional examples.

Static mapping response systems use decision tables to directly map an attack scenario to hard-coded response. Such approach is context-independent as it fails to take into consideration the unique circumstances under which the attack was triggered. For example, taking the system offline is a possible response to any intrusion, but if a system in question is the mail server of a large organization, taking it offline may prove to have disastrous consequences. If on the other hand, a backup server exists which can perform the same duties, then disconnecting the primary server in case it was compromised may not turn out to be such a bad idea.

Dynamic mapping response systems respond to an attack by considering circumstantial evidence surrounding the attack. Such systems are implemented as rule-based decision engines and expert systems. Each rule is a nested hierarchy of implications that decide actions based on the following factors [17,25,37]:

- Type of attack (probing, denial of service)
- Confidence level (how many monitored features substantiate that the attack is taking place?)
- Incident severity
- Reliability of intrusion detector that triggered the alarm (may depend on false positive rate)
- The utility of the target system for users and the organization

- Type of perpetrator (script kiddie vs. professional cracker, external party or insider)
- Privileges associated with the compromised user account (if applicable)

Expert perpetrators are persistent and resourceful; if their attack fails initially, they will adapt their tactic so as to avoid the defense and try again. So the response system itself must adapt its strategy.

Chapter 5

NIDS for Distributed Networks

Most commercial intrusion detection architectures suffer from their lack of ability to scale well to large networking environments. They tend to have centralized audit collection and storage mechanisms that place heavy burden on available computational resources. As networks grow in size by increasing the number of individual devices and adding new services, it becomes increasingly difficult to manage the ever-growing knowledge repositories. The statistical analysis engines as well as signature detection engines have to sort through massive amounts of audit data in order to identify anomalous or malicious behavior. As a result performance substantially declines, attacks are failed to be detected in real time due to overhead in data processing, the instance of false alarms rises due to excessive noise. In addition, the centralization of intrusion detection provides a single point of failure. This chapter introduces two network-based commercial intrusion detection systems that can scale to large enterprise networks.

5.1 EMERALD

The EMERALD project introduces a distributed, scalable, extensible, interoperable intrusion detection system designed for large enterprise networks [16]. EMERALD provides a complementary analysis over the operation of networks and individual

entities within them by combining signature-based detection and statistical profiling to detect anomalies. At the heart of EMERALD IDS is a layered hierarchy that divides the task of network surveillance among three different types of monitors. Service monitors are responsible for individual network components and services within a single domain. Domain monitors handle intrusion detection in the context of entire domain. Enterprise monitors provide global protection across all monitored domains and facilitate the detection of network-wide threats such as internet worms and large coordinated attacks. Certain types of intrusions manifest globally and are not made visible by monitoring individual network assets.

Data collected on the lower levels of the surveillance hierarchy is transmitted to upper levels by a subscription mechanism. Service monitors collect audit data from individual components and compile analysis results which then get propagated "up" to domain monitor, which in turn performs correlated analysis of data received from all its service monitors. Analysis reports from multiple domain monitors are then sent to an enterprise-level monitor. Such hierarchical approach to intrusion detection does not impose heavy burden on computational resources because the responsibility of data collection and analysis is divided among multiple monitors. This facilitates timely detection and consequently, early response to threats. Furthermore, since each monitor is responsible for a data stream belonging to each individual network component, it is easier to filter out noise and perform a more focused and fine-grained data analysis. For example, each monitor needs to maintain only those signatures and statistical profiles which are relevant to the target being monitored. A target could be anything: a host, a service, a router, a domain, an entire network, etc.

5.1.1 EMERALD Monitor Architecture

Each monitor, regardless of its scope (service, domain or enterprise) has four essential components: profiler engine, signature engine, resolver and resource object [16].

Profiler engine performs statistical analysis on the event stream in order to classify anomalous activity. Modularity is achieved by logically separating profile management from statistical framework for anomaly scoring. EMERALD Signature engine employs a rule-based inference signature detection scheme. Each individual knowledge base is specifically tailored for a single analysis target. This reduces noise ratio, increases accuracy of detection and improves the overall performance.

Resolver is EMERALD's countermeasure expert system. It processes intrusion and anomaly reports, implements an appropriate response policy, manages internal analysis engines and interfaces with external engines through a subscription service in order to correlate analysis results and disseminate intrusion reports to other monitors. Response policy is determined by inferring the confidence level of the attack and the level of its severity. Resolver could respond by generating an intrusion alert, closing a connection, terminating a process, etc.

Resource object is a pluggable library that contains target-specific data structures, functions and configuration variables. This component is what differentiates one monitor from the next. Both the resolver and analysis engines are independent of the target on which the monitor is deployed. Resource object specifies data format, analysis results syntax and semantics, statistical thresholds, intrusion signatures, filtering routines that convert target-unique events into general form, subscription lists, response handlers, etc.

5.2 GrIDS

GrIDS is a real-time graph-based network intrusion detection system [29]. It is designed to scale well to large network environments and provide detection capabilities against large-scale coordinated or automated attacks, especially internet worms. GrIDS views a network as a collection of users, hosts and departments that communicate via pair-wise network connections defined by their application protocol (HTTP, TELNET, etc). Each department itself is a collection of others departments, users and hosts. Attack scenarios are represented as *activity graphs*.

Activity graphs define traffic patterns among individual network entities over a given period of time. Each graph consists of nodes and directed edges. Nodes represent hosts or departments, while edges represent network traffic between them. For example when host A initiates connection to host B, an edge leading from A to B is generated. Different attacks generate characteristic types of graphs. For example, worms are malicious programs that propagate across the network by invading one machine and then using its resources to invade its neighbors [29]. Traffic produced by worm propagation forms a tree-like activity graph. Example graph is illustrated in figure 8.

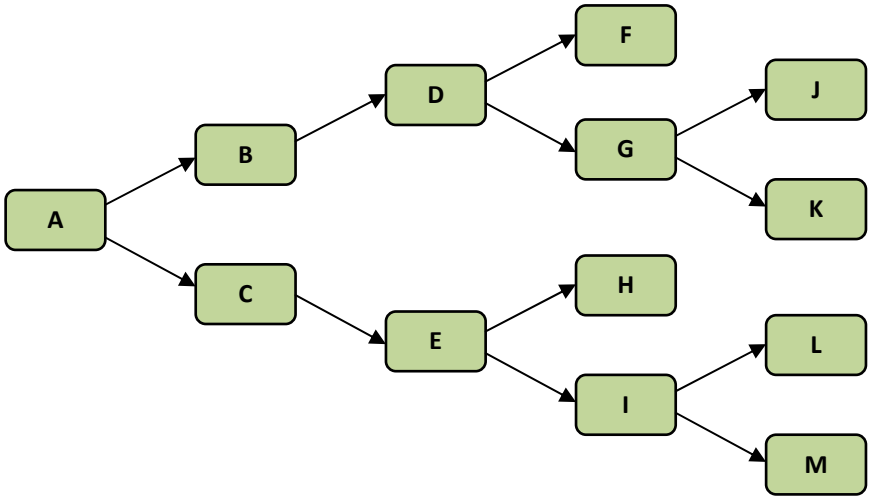

Figure 8: Worm activity graph

Nodes, edges and graphs are defined with supplementary attributes. Certain attributes of the graph, in particular size, depth and branching factor, can be used to make inferences about the confidence or severity of the attack. GrIDS, like EMERALD, features a multi-layer surveillance hierarchy. Each department aggregates hosts as well as other departments and has its own graph-building engine, which evaluates activity graphs within the department. In order to model activity among departments on the same level of hierarchy, those departments are reduced into a single node. Figure 9 demonstrated how departments are viewed from a higher level. The topology of reduced departments is lost at least as far as higher level module is concerned, but their attributes (size, depth and branching factor) which do get passed "up", are much more relevant for the purposes of identifying intrusions.

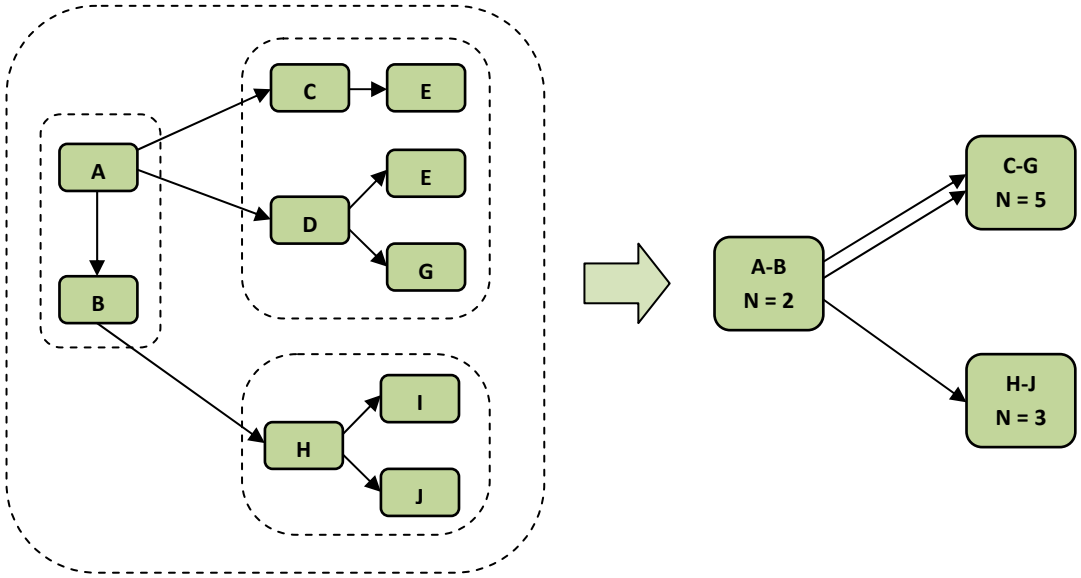

Figure 9: Reducing departments into a single node

GrIDS data sources are modules that monitor activity on individual hosts and networks and send activity reports, in a form of a discovered node or new edge, to the graph engine which then incorporates the new information into its active graphs. Modules come in a form of packet sniffers or external NIDS systems spread around the network. When the engine on the lower level builds a graph, it generates a summary and passes it to the engine for its parents department, which will in turn incorporate new information into its own graph.

Graphs are generated in accordance to user-specified rule sets. A rule set is a formal specification of one kind of graph. Rules specify conditions under which an activity report will be incorporated into a graph structure, thresholds that determine when the graph should be treated as a sign of intrusive activity and which actions to take in response, as well as conditions under which overlapping graphs can be combined.

Chapter 6

Efficiency Evaluation

In order to be an effective and reliable security solution, intrusion detection system must fulfill certain criteria. In order to assess how intrusion detection system performs on a given network, the following metrics are often considered [4]:

- *Accuracy of detection*: ideally intrusion detection system would flag all illegitimate behavior as intrusive and not alarm administrators when detecting behavior which is characteristically different from normal, but nonetheless legitimate. False positive occurs when IDS reports an intrusion when there is none. False negative occurs when an intrusion fails to be reported.
- *Completeness*: it is desirable, though rarely achievable in practice, to design NIDS that successfully detects all attacks present in network traffic. Since perfect completeness cannot be reached, we quantify how closely NIDS approaches this ideal mark.
- *Performance* measures how effectively IDS processes audit data. Poor performance results in delayed attack detection. It is important to detect all intrusions in real-time before any serious damage is done to the network. Delayed detection results often results in significant loses.
- *Fault tolerance* refers to the ability of IDS to resist attacks against itself. A network where IDS itself is vulnerable to attacks is in many instances as

insecure as a network with no IDS at all. For example, if a skilled attacker locates unprotected knowledge repository that contains attack signatures, he could delete signatures that detect specific attacks and then carry out those attacks, avoiding any detection.

One class of attacks against IDS is called "crash attacks" [3]. Such attacks attempt to undermine the operation of IDS by causing it to crash or enter a processing jam. Fault tolerant system has mechanisms in place that minimize the risk of being disabled by the attacker. For example, Bro is a real-time network intrusion detection system that implements a fault-resistant mechanism known as the "watchdog" timer. This timer is activated when Bro begins processing a new event. If the system is still processing the same event when the timer expires, it assumes to be in a processing loop so it drops current event and proceeds to analyze the next one.

Certain Denial of Service attacks specifically target intrusion detection systems. This is accomplished by generating a large quantity of audit data and feeding it directly into the event monitor of the IDS. If enough data is generated this way, the analysis engine will be overwhelmed and real-time detection will be effectively disabled. One possible solution [3] is to alleviate the load by discarding data flows for which the state is maintained, yet no real progress is made. In essence, the system is making a decision to ignore data which is assumed to have low priority in order to effectively process new data.

6.1 Benchmarking

The behavior of each network is a function of its operational domain (research enterprise, educational enterprise, commercial enterprise, business enterprise, etc), its users (their roles, experience level, habits, etc), applications (e-commerce, gaming, www, streaming video, etc). Each network environment has distinct characteristics that might affect the performance of anomaly detection systems deployed on it. These characteristics include traffic profiles (UDP vs. TCP), traffic volume, number of connection requests per second, relative volume of incoming and outgoing traffic, most frequently used applications (HTTP, FTP, Mail), etc.

6.1.1 Case Study: Benchmarking Based on Data Entropy

For anomaly detection systems that implement statistical modeling, one characteristic of particular importance is data entropy or irregularity. The availability of data that is highly regular or redundant facilitates accurate prediction of future events based on past events. In one benchmarking study [15], the effects of data entropy on accuracy of anomaly detection were evaluated and the following conclusions were reached: (1) differences in data regularity do influence the performance of the anomaly detector and (2) such differences can be found in natural environments. The practical implication of these conclusions is that performance of anomaly detection system cannot be evaluated by running it on datasets of the same regularity level, since there is strong evidence to suggest that running the same detector on a dataset of different regularity will generate a different performance index.

Shifting regularities persist across domains, across networks and across individual users. Intrinsic structure of audit data used by anomaly detector affects its performance. One possible solution is to monitor level of regularity in real time and swap current anomaly detector with another if it is expected to perform better for new regularity level. Another solution is to use an adaptive anomaly detector that automatically calibrates its detection parameters in response to shifts in regularity. One possible parameter is the anomaly threshold that determines the magnitude about which the anomaly is taken seriously (usually measured on 0-1 scale).

In order to understand the notion of data entropy, consider a simple audit data sequence where each event is an action taken by a user: system login (A), mail check (B), open file spreadsheet1.xls (C) and logout (D). If a behavior of a given user is examined over a period of days, the following data sequence might be generated: ABCDABCD… Some events in such sequence necessarily precede other events. Such sequential dependency between events in a categorical data is accounted for by a measure called *conditional relative entropy*. Relativity reflects the fact that entropy does not have upper bound, yet it still needs to be measured on a fixed scale. Conditionality reflects the fact that both the probability of event and its predecessor needs to be accounted for. Regularity index 0 signifies perfect regularity (redundancy) while regularity index 1 signifies perfect randomness.

The following types of anomalies were introduced into testing data sets: foreign symbol anomalies, foreign n-gram anomalies and rare n-gram anomalies.

- *Foreign symbol anomaly*: this is the simplest type of anomaly. If the training data does not contain a certain kind of symbol (event), the occurrence of

such event would constitute a foreign symbol anomaly. If the training data contained only symbols A, B, C and D, symbol E in testing data would be considered anomalous.

- *Foreign n-gram anomaly*: if the training data is generated using an alphabet of symbols (events) of size a, then there are a^n possible n-grams of size n. If a certain n-gram (sequence of events) does not appear anywhere in the training data, then the occurrence of such n-gram would constitute a foreign n-gram anomaly. In the example given earlier bigram CB would be considered anomalous because event C always follows event B.
- *Rare n-gram anomaly*: n-grams that appear infrequently in the training data are considered rare. Rarity is determined by a user-specified threshold.

When the anomaly detector runs on a dataset in training mode, it creates a profile of what is considered normal behavior. The output of training session consists of specialized data structures needed to detect anomalies. Specifically, it constructs an internal table that for each unique n-gram stores the probability of its occurrence. During testing session, detector is run on datasets containing randomly injected anomalies. In order to determine the effects of data regularity on the detector performance, both training and testing were done on data of the same regularity index. The performance of the detector was evaluated with respect to accuracy of detection.

The experimental results [15] demonstrated a strong correlation between regularity levels and false-alarm rate. In particular, false-alarm rate increases rapidly as the regularity index grows. When regularity index reaches 0.8, performance is degraded significantly and for completely random data, false alarms reach 100%. In the second

experiment, regularities of system-call streams of 58 users were recorded. Considerable deviations were identified, proving that data regularity is a real-life phenomenon bearing both theoretical and practical significance for the field of anomaly detection.

Chapter 7

Related Works

This chapter presents several technologies and areas of research that take the idea of information security into a new direction. It should be noted that these solutions are not alternatives to intrusion detection. Instead they complement and extend NIDS in new interesting ways.

7.1 Honeypots

Historically, the major paradigm of information security has been defense. Intrusion detection systems, along with firewalls and encryption technologies have been primarily focused on protecting valuable assets from theft, corruption and authorized access. As networks grow in size and complexity, the number of vulnerabilities increases exponentially and so does the incidence of attacks. Signature detection methods are becoming less and less effective due to increasing size of knowledge repositories that need to be constantly updated and maintained as well as polymorphic attacks, mentioned in chapter 1. Anomaly detection engines are becoming more popular due to their inherent ability to detect novel attacks. But there is an associated cost of high rate of false alarms.

There is a strong indication that the incidence rate of attacks will continue to increase, while tools and techniques used by hackers will grow in sophistication and

variability. It is becoming increasingly difficult for security professionals to keep up with the enemy who has the upper hand. In many instances a new attack gets discovered only after the damage is already done and the system has been compromised. Even if an anomaly engine manages to detect anomalous traffic associated with the attack, the implementation of the right response policy is rarely a clear-cut decision.

The assessment of countermeasures depends on the availability of additional information, such as the identity of attackers, how they got in, how much damage they have caused, etc. Since the attack is new, this information is rarely obtainable within a short timeframe. The response and recovery effort is further impeded by the fact that certain systems, even if compromised, cannot be taken offline due to the critical functionality they provide. For example, the mail server of a large organization is a crucial production asset and making it unavailable even for a short period of time might cause more damage than attackers ever hoped for.

Part of the problem, as has been previously mentioned, is that the strategy for dealing with intruders has been primarily defensive. Once the network is armed with firewalls, intrusion detection systems, latest signatures, traffic monitors, it enters a waiting state hoping that the defenses withhold the attack long enough for the right countermeasures to be devised or detect it early enough so that no damage is made. A technology, called honeypot attempts to change that.

Put simply, a *honeypot* is an information system resource whose value lies in its unauthorized or illicit use [23]. Honeypot is a dummy system with no production value. Therefore it sees no legitimate traffic and any interaction with a honeypot is an

indication of malicious activity. The purpose of honeypots is to lure hackers into attacking them with the intent of gathering information. Every action performed by the intruder, from viewing files to individual keystrokes, can be logged and later analyzed. Unlike a production system which potentially sees hundreds of users and logs gigabytes of daily activity, honeypot collects small amount of very valuable information. Since all activity is malicious, there is no noise and data analysis is both cheap and easy.

Like all technologies, honeypots have associated risks. It should already be clear that honeypot loses its value as soon as the attacker who is interacting with it becomes aware of its presence. The danger however lies in the fact that some skilled hackers could use a compromised honeypot to lunch further attacks against other systems. Second, there is a risk that once attackers identify a honeypot, they will purposefully perform bogus actions in order to mislead data analysts, or worse, they could locate data capturing facilities and directly inject false information. They could also covertly disable specific parts of honeypot functionality.

Honeypots fall into two general categories: low-interaction honeypots and high-interaction honeypots [23]. *Low-interaction honeypots* emulate services and operating systems. Each emulated service supports a subset of functionality that the actual service allows. For example, emulated FTP service may support login and only few other commands. The main advantage of low-interaction honeypot is that it is easy to deploy and maintain. Also, since emulated services allow limited functionality, the risk of a honeypot being turned into a hacking asset is substantially reduced. The disadvantages include narrow information gathering capabilities, limited mainly to known attacks, and high probability of exposure.

Honeyd is most widely used low-interaction honeypot system. It monitors the space of unused IP address on a network, and intercepts and redirects any connection attempt to such IP address to itself, pretending to be a victim computer. It then carefully logs all the activity including all inputs and all issued commands. Honeyd can emulate services as well as entire operating systems (over 500) [22], both at the IP stack and application level. This way, if hackers attempt to use OS fingerprinting tools, Honeyd will issue a response in accordance with the design of IP stack of the OS it is emulating. Furthermore, Honeyd is not limited to emulating only desktop operating systems. For example, it could just as easily pretend to be a Cisco router. Honeyd is a very powerful tool that can be used to build an entire network of virtual devices, each running its own operating system and services [23].

High-interaction honeypots involve real operating system and services. There is no emulation of any kind. While deployment and maintenance of such systems is without a doubt costly endeavor, the gains in power outweigh the costs. High-interaction honeypots allow attackers to utilize full functionality of a system and therefore log extensive amount of information. Data collection is facilitated by a special activity-logging kernel module. This allows security professionals to learn the full extent of hackers' behavior, identify new tools they use and gain insight into their motives.

7.1.1 Honeynets

The concept of a high-interaction honeypot could be extended to a *honeynet* [21]. A honeynet is an entire network of honeypots. Within this network all activity is carefully

controlled and logged. Once again, any incoming traffic into a honeynet is assumed to be malicious. At the heart of the honeynet architecture is a *honeywall*, a gateway device that separate a honeynet from the rest of your network as well as the internet (See figure 10). Honeywall is layer-2 bridge. Since bridges are transparent to protocols above the MAC layer, they are invisible to anyone interacting with a honeynet. Honeywall is responsible for monitoring and logging all incoming traffic into a honeynet. This data is stored on an external secure system in order to ensure it's not detected by hackers who might make modifications to it or delete it all together. After the data has been collected, it needs to be analyzed and converted into applicable form.

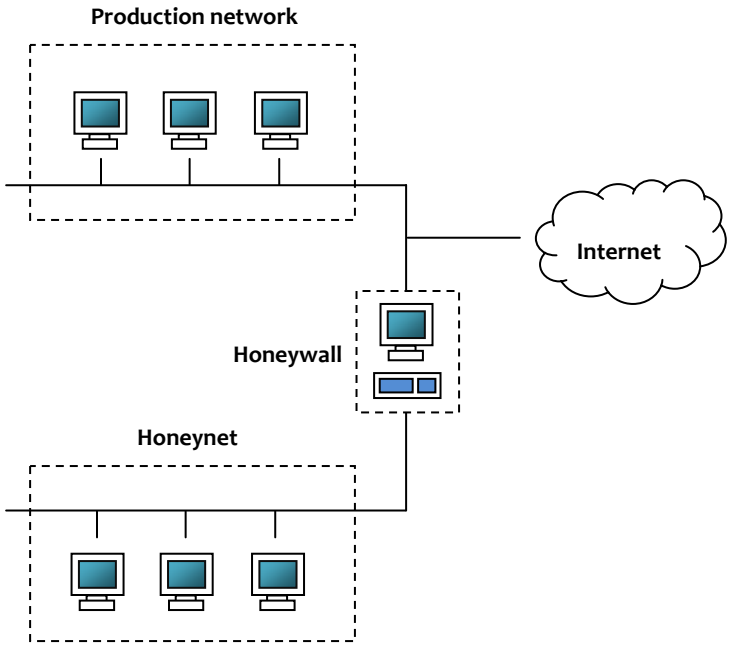

Figure 10: Honeynet architecture

The final responsibility of a honeywall is Data Control: isolation of a honeynet from the rest of the network. Every effort must be made to ensure that once the attackers enter the boundary of the honeynet, they are denied access to any outside

system. Data Control acts as an intrusion prevention gateway and firewall. Generally, the more freedom attackers are allowed to have within a honeynet, the more can be learned about them, and the higher the risk that Data Control will be evaded.

When used for research purposes, honeypots collect information on hacker's activities, specific tools and techniques they employ in order to probe networks, exploit vulnerabilities and gain unauthorized access to victim computers. This information could then be used to make systems more secure by fixing specific vulnerabilities that allowed hackers to get in, craft new signatures for intrusion detection systems or simply study security trends. Most importantly, honeypots allow security professionals to devise response strategies against new attacks before they even occur on production systems.

Honeypots can also be used by organizations to provide real-time protection against intrusions. Network scanning utilities probe the target network by scanning its IP space for live hosts and then attempt to connect to open ports. By introducing a honeypot that monitors unused IP space for connection attempts, network probes can be intercepted and slowed down. Such honeypots are known as *sticky honeypots*. As a concrete example, `LaBrea` is a program that creates a virtual machine for each unused IP address on a network [23]. Each virtual machine waits for a connection attempt and when one is made, it uses various TCP techniques to cause the sender, in our case a network probe or even a worm, to get "stuck", sometimes for a long time. One strategy is to set window size of the response packet to zero, effectively placing sender into a holding pattern as he waits for the window size to increase so he may send data.

Another way honeypots can protect networks from intrusions is by confusing and slowing down human attackers. While intruders are wasting their time interacting with honeypots, it gives enough time to network staff to respond and circumvent the attack. Honeypots are an extremely valuable security asset. By capturing small data sets of high value, they facilitate the detection of new exploits and even polymorphic shellcode attacks discussed in chapter 1. In a way honeypots are an ultimate form of anomaly detection, because any activity within a honeypot is guaranteed to be an anomaly.

7.1.2 Dynamic Honeypots

Networks are dynamic systems: new devices are added and removed, operating systems are updated, new applications and services are constantly introduced. As the networks change, the honeypots themselves need to be modified to reflect the new network infrastructure. Outdated honeypots quickly lose their value as both information-gathering and intrusion prevention assets. Furthermore, if honeypots and honeynets don't mirror their production environment, they would stand out too much and become easily identifiable. A traditional solution to this problem is manual reconfiguration of honeypots by network administrative staff. This means time, money and unavoidable mistakes.

A new proposed solution is a *dynamic honeypot* [22] that automatically analyzes the network on which it is deployed and adapts to it. The idea is to use real-time passive fingerprinting to determine which operating systems are currently used in your network, how many of each type there are, the IP range on which each type of systems

resides on, and which services they run, and then have a system like Honeyd mirror your network.

7.2 Honeycomb

Generating intrusion detection signatures is an arduous and tedious task, generally requiring extensive knowledge and expertise of security professionals. As was discussed in chapter 2, many different standards exist for translating an intrusion scenario into a formal signature. Since signature languages of various NIDS differ in both syntax and level of expressiveness, the signatures cannot be easily ported from one engine to be used in the next. These issues can be addressed by automatic signature generation. We will use Honeycomb [12] as a case study to describe how network anomalies can be transformed into signatures representing various attack scenarios.

Honeycomb is a perfect example of how different security technologies and concepts can extend each other's functionality in new interesting ways. Honeycomb is implemented as a honeyd plug-in that has the following components: protocol analyzer, flow reassembler, pattern matcher and signature generator. Network traffic at the honeyd honeypot is inspected at the network and transport layers. For each packet flow, an empty signature is created and continually augmented with new *facts*. Flow here refers to a stream of packets with the same source/destination IP and port.

Packet analyzer examines TCP, UDP and IP packet headers and attempts to detect anomalies, for example unusual TCP flag combinations. Each discovered anomaly is recorded as a new fact in a signature. The flow reassembler collects all packets

belonging to the current flow. Pattern matcher then attempts to find similarities in packet payloads between a current connection's flow and those for which connection state is continually maintained. Each matched pattern is treated as a new fact and added to the signature.

Periodically, the signature pool is examined and duplicate signatures are dropped while related signatures are aggregated. Signature generator translates native signatures into a specific format accepted by NIDS. Currently, `Honeycomb` generates signatures for `Bro` and `Snort` intrusion detection systems. Initial testing showed promising results: high-quality signatures were generated for `CodeRed II` and `Slammer` worms and various port scanners [12].

7.3 Identifying the Source of DoS Attacks

In chapter 1, we discussed how entire networks can be brought down by flooding. We also mentioned that there are two typical ways of dealing with DoS attacks once they have been detected: restoring the network operation by countering the effects of flooding or identifying the source of the attack. Identifying the source of DoS attack is a lot more difficult that may initially seem since most DoS attacks use source IP spoofing and don't leave much trace. However, one research proposes an interesting alternate solution [10].

The idea is to reprogram all routers to "stamp" each packet with a special meta-data that uniquely identifies the router. This way upon receipt, one could extract all collected meta-data and theoretically trace the packet all the way to its source – the attacker himself. Such solution could put an end to DoS attacks all together. If attackers

knew that there was no way to conceal their identity, it would be unlikely for them to proceed with the attack in the first place.

There are still several issues with the proposed solution which need to be addressed. The most intuitive way to identify a router is by its internal IP address. But if all routers stamped all packets with a 32-bit IP address, then by the time packets reached their destination, the meta-information they would accumulate along the way would introduce a substantial transmission and queuing delays which would undermine the effective operation of the entire internet.

The solution is to store only a portion of the entire route inside each packet. Each router would randomly decide based on some hard-coded probability whether to stamp a given packet with its IP address as well as distance from the source of the route. Luckily DoS attacks generate a substantial amount of traffic in order to accomplish their goals, so chances are that every router would stamp at least some packet with this information. At the destination, a sequence of packets will be examined and the entire route will be reconstructed piece by piece.

Even better solution is to stamp packets with smaller 16-bit AS (autonomous system) numbers belonging to entire routing domains, as oppose to individual routers. Such numbers could be easily stored inside unused 16-bit IP fragmentation field. However, skilled hackers could intentionally use fragmentation field by fragmenting packets they send into the network. One solution that could be realized in the near future is to reserve a dedicated 16-bit field for AS sampling in the upcoming IP protocol implementation (IPv6).

Conclusion

Networks are becoming increasingly complex as organizations add new applications, devices and users. As the value of our network infrastructure continues to grow, so does our need to ensure that these assets are adequately protected against attacks that proliferate in diversity, sophistication, speed and incidence. We have reviewed four types of systems: signature detection systems, anomaly detection systems, intrusion response systems and honeypots. No single type of system has the potential to effectively undertake the task of network surveillance all by itself. There is an elevated need for integrated solutions that rip the benefits of both kinds of detection paradigms, coupled with a flexible autonomous response element. The overhead of configuring and maintaining multiple systems impedes this effort especially in the context of networks that are highly dynamic and unpredictable. Therefore we need extensible systems that adapt to changing conditions as well as interoperate with other systems to provide complementary lines of defense. We believe that the future of network surveillance will be driven by design methodologies we have just identified: interoperability, adaptability, extensibility and scalability.

References

[1] O. Arkin. Network Scanning Techniques - Understanding How it is Done. *PubliCom Communication Solutions*, Nov. 1999

[2] S. Axelsson. Intrusion detection systems: A survey and taxonomy. *Technical Report*, Chalmers Univ., March. 2000

[3] D. J. Brown, B. Suckow, and T. Wang. A Survey of Intrusion Detection Systems. Department of Computer Science, University of California, San Diego

[4] H. Debar. Introduction to Intrusion-Detection Systems. *IBM Research*, Zurich Research Laboratory.

[5] D. Denning. An Intrusion-Detection Model. *IEEE Trans. on Software Eng*., February 1987

[6] L. Deri, S. Suin and G. Maselli. Design and implementation of an anomaly detection system: An empirical approach. *In Proceedings of Terena TNC*, 2003

[7] P. García-Teodoro, J. E. Díaz-Verdejo, G. Maciá-Fernández, and L. Sánchez-Casad. Network-based Hybrid Intrusion Detection and Honeysystems as Active Reaction Schemes. *IJCSNS International Journal of Computer Science and Network Security*, VOL.7 No.10, October 2007

[8] Dr. F. Gong. Deciphering Detection Techniques: Part II Anomaly-Based Intrusion Detection [whitepaper]. *McAfee Network Security Technologies Group*, March 2003

[9] Dr. F. Gong. Deciphering Detection Techniques: Part III Denial of Service Detection [whitepaper]. *McAfee Network Security Technologies Group*, January 2003

[10] M. R. Hines. Going Beyond Behavior-Based Intrusion Detection. Dept. of Computer Science, Binghamton University, Fall 2003

[11] A. K. Jones and R. S. Sielken. Computer System Intrusion Detection: A Survey. Department of Computer Science, University of Virginia, 2000

[12] C. Kreibich, and J. Crowcroft. Honeycomb - Creating Intrusion Detection Signatures Using Honeypots. *HotNets-II*, Cambridge, USA, November 2003

[13] S. Kumar and E. H. Spafford. A pattern matching model for misuse intrusion detection. *In 17th National Computer Security Conference*, 1994.

[14] K. Leung and C. Leckie. Unsupervised Anomaly Detection in Network Intrusion Detection Using Clusters. *Proc. 28th Australasian CS Conf*

[15] R. A. Maxion and K. M. C. Tan. Benchmarking anomaly-based detection systems. *In Proceedings of 2000 International Conference on Dependable Systems and Networks*

[16] Dr. P. Neumann and P. Porras. EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances. *1997 National Information Systems Security Conference*, October 1997

[17] K. Schafer. Intrusion Response – The Right Course of Action. University of Idaho

[18] R. Sekar, A. Gupta, J. Frullo, T. Shanbhag, S. Zhou, A. Tiwari and H. Yang. Specification-based Anomaly Detection: A New Approach for Detecting Network Intrusions. *ACM CCS*, 2002

[19] V. A. Siris and F. Papagalou. Application of anomaly detection algorithms for detecting SYN flooding attacks. *Global Telecommunications Conference*, 2004. IEEE

[20] Y. Song, M. E. Locasto, A. Stavrou, A. D. Keromytis and S. J. Stolfo. On the Infeasibility of Modeling Polymorphic Shellcode. *In Proceedings of the 14th ACM Conference on Computer and Communications Security* (CCS). October 2007, Alexandria, Virginia, USA

[21] L. Spitzner. Know Your Enemy: Honeynets. *Honeynet Project*, May 2006

[22] L. Spitzner. Dynamic Honeypots. *Security Focus*, September 2003

[23] L. Spitzner. Honeypots: Definitions and Values. *Honeypots: Tracking Hackers*, May 2003

[24] L. Spitzner. Know Your Enemy: Passive Fingerprinting. *Honeynet Project*, March 2002

[25] N. Stakhanova, S. Basu, J. Wong. A Taxonomy of Intrusion Response Systems. Department of Computer Science, Iowa State University

[26] K. Takemori, K. Rikitake, Y. Miyake, and K. Nakao. Intrusion Trap System: An Efficient Platform for Gathering Intrusion-related Information. *Proceedings of the 10th International Conference on Telecommunications* (ICT2003), IEEE, 2003

[27] M. Thottan and C. Ji. Anomaly Detection in IP Networks. *IEEE Transactions On Signal Processing*, Vol. 51, No. 8, August 2003

[28] M. Roesch. Snort – Lightweight Intrusion Detection for Networks. *Proc. USENIX LISA '99 Conf.*, Nov. 1999

[29] S. Staniford-Chen, S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland, K Levitt, C. Wee, R. Yip, and D. Zerkle. GrIDS – A graph based intrusion detection system for large networks. *In Proceedings of the 19th National Information Systems Security Conference*, 1996

[30] K2. ADMmutate README. ADMmutate source code distribution. Version 0.8.4. URL: http://www.ktwo.ca/c/ADMmutate-0.8.4.tar.gz (Jan 2002)

[31] E. J. Bowden. Network-based intrusion detection system buyer's guide [whitepaper]. *Echo Identity Systems, Inc*.

[32] L. A. Gordon, M. P. Leob, W. Lucyshyn, and R. Richardson. CSI/FBI Computer Crime and Security Survey, 2005.

[33] M. Tanase. IP Spoofing: An Introduction. Security Focus, March 11, 2003.

[34] S. M. Specht and R. B. Lee. Distributed Denial of Service: Taxonomies of Attacks, Tools, and Countermeasures. *In Proceedings of 17th International Conference on Parallel and Distributed Computing Systems*, *2004 International Workshop on* Security in Parallel and Distributed Systems. September 2004.

[35] G. Münz, S. Li and G. Carle. Traffic Anomaly Detection Using K-Means Clustering. University of Tuebingen, Germany, 2007.

[36] C. Carver, J. M. Hill, and J. R. Surdu. A methodology for using intelligent agents to provide automated intrusion response. *In Proceedings of the IEEE Systems, Man, and Cybernetics Information Assurance and Security Workshop*, 2000.

[37] Dr. S. Furnell and M. Papadaki. Automated Intrusion Response. Network Research Group, School of Computing, Communications & Electronics, University of Plymouth.

[38] CERT. Cert statistics: Vulnerability remediation. CERT Web Site, April 2008. http://www.cert.org/stats/vulnerability_remediation.html.