Defeating Internet Attacks Using Risk Awareness and Active Honeypots

Lawrence Teo[1,2] Yu-An Sun[1] Gail-Joon Ahn[1]

[1] *Laboratory of Information Integration, Security and Privacy (LIISP), University of North Carolina at Charlotte, 9201 University City Blvd, Charlotte, NC 28223, USA.*

[2] *Calyptix Security Corporation P.O. Box 561508, Charlotte, NC 28256, USA. http://www.calyptix.com/*

{*lcteo,ysun,gahn*}*@uncc.edu*

Abstract

New forms of Internet attacks, such as SQL Slammer, have become increasingly sophisticated. Although coded in a simple way, the SQL Slammer worm propagated all over the world at an extremely high speed in a short period of time, rendering it impossible for humans to counter it using manual intervention. In this paper, we propose a security framework called Japonica to detect and respond to unknown attacks at the early stage through the dynamic orchestration of prevention, detection, and response mechanisms. We identify important requirements to support the proposed framework and corresponding system entities. Also, we describe our model using Colored Petri Nets to discover a uniform message exchange format among the entities. One unique characteristic of Japonica is an active response coordinator and we demonstrate its feasibility in a proof-of-concept prototype, utilizing a honeypot as an active entity. Our results indicate that Japonica can successfully prevent the spread of SQL Slammer without human intervention. We are currently extending the framework to counter other forms of sophisticated Internet attacks.

Keywords: Japonica, Honeypots, Risk Awareness, Colored Petri Nets.

1. Introduction

Many organizations today use firewalls and intrusion detection systems (IDSs) as part of their network security defenses. Apart from these two technologies which are now commonly used, a honeypot has received much attention in recent years. A honeypot can be thought of as a decoy computer system that uses deception to lure intruders so that we can learn their behaviors. The honeypot is usually a system that is deliberately made vulnerable with fake services to make it look and act like a real system. Intruders who discover the honeypot may choose to compromise it since it is a relatively easy task. As a result, system administrators can investigate the traces left by intruders to learn about their tools and techniques in detail.

In this paper, we describe a novel approach to use honeypots as an *active component* to defend a local network, instead of using it as a passive component. We attempt to use a honeypot to complement the functionality of a network IDS. By allowing the honeypot to coordinate its work proactively with the firewall and the IDS, we can achieve early response to network security incidents. Early response to such incidents is very important, as Internet attacks become increasingly sophisticated and fast. We believe early response is a critical requirement. For example, the recent SQL Slammer worm [17] is the first high-speed Internet worm that propagated around the Internet in just ten minutes. SQL Slammer showed that automatic defense mechanisms are needed to counter future unknown threats at the early stage, and manual human intervention is no longer feasible [31].

Our approach of using honeypots as an active component is inspired by an interesting natural phenomenon that is demonstrated by how Japanese honeybees defend their beehives against mass attacks by giant hornets [13, 21, 22, 27]. To understand such a strategy, we elaborate how the giant hornets attack a hive belonging to regular honeybees (not the Japanese bees). First, a hornet leaves its pheromone at the hive's entrance. The pheromone attracts other hornets, which causes them to arrive and attack the hive together. Since the honeybees' stingers cannot penetrate the hornets' armor, the result is a mass slaughter of the honeybees, where 30-40 hornets can kill 30,000 honeybees in just a few hours.

Japanese honeybees, however, employ an unusual defense strategy to counter the hornet's attack. When a hornet marks the hive's entrance, the honeybees guarding the en-

Proceedings of the Second IEEE International Information Assurance Workshop (IWIA'04)
0-7695-2117-7/04 $ 20.00 © 2004 IEEE

trance would return to the hive. This lures the hornet *into* the hive, rather than allowing them to start their attack outside. At the same time, over 1,000 worker honeybees would leave their combs and form a mass just inside the hive's entrance. When a hornet tries to enter the hive, the bees surround it and form a ball of bodies, and as a result, they cook the hornet to death with their thermal body heat. By killing the first few hornets, they prevent the hornet's pheromone from attracting more hornets. Thus, their hive is protected by invoking early response to the hornet's attacks.

In the context of our research, the hornets represent intruders, while the bees represent our coordinated early response strategy based on honeypot, firewall and IDS technologies. The hornet's initial attack can be viewed as early network reconnaissance (e.g. port scans) or the beginning of a high-speed attack such as distributed denial-of-service (DDoS) attacks or Internet worms. The honeybee's thermal defense strategy is akin to our early response approach: by detecting and responding to such attacks at the first place, we can prevent the attacks from growing and wreaking further damage to the network.

Using a honeypot as an active component for early response has significant advantages. First of all, by complementing the functionality of a network IDS, there would be no false positives since only intruders would access the honeypot but not legitimate users. Secondly, by orchestrating the honeypot to work in conjunction with an IDS and firewall, we can achieve dynamic and automatic early response against attacks. Finally, we can use the honeypot to provide information to the IDS and firewall about attacks that are local and specific to the organization's environment.

In this paper, we propose a security framework called Japonica, which uses the honeypot as an active component to work together with prevention and detection mechanisms such as an IDS and firewall in order to implement an organization's network defense mechanism. By leveraging the dynamic orchestration of the prevention, detection, and response mechanisms in this manner, we can achieve early response to sophisticated attacks such as SQL Slammer more accurately.

The rest of the paper is organized as follows. Section 2 describes the relevant background and related work. Section 3 presents the proposed Japonica framework. Section 4 details a proof-of-concept experiment utilizing a Japonica implementation. Section 5 discusses lessons learned as well as the strengths and limitations of Japonica. Finally, we provide the conclusion in Section 6.

2. Background and related work

In this section, we will discuss the background of honeypots and risk-aware defenses that are relevant to our framework.

2.1. Honeypots

Despite recent interest, the concept of honeypots was actually conceived many years ago. The first extensively documented honeypot was described by Stoll in his book The Cuckoo's Egg [32, 33] (although it was not yet called a honeypot at that time). Cheswick and Bellovin were also instrumental in describing how honeypots could be used to learn about attackers [6]. The purpose of such honeypots then was to confine or contain the intruders as long as possible to distract them from attacking production systems.

Other related work include Cohen's deception tool kit [8] and Spitzner's work on a distributed network of honeypots, which is known as a honeynet. The Honeynet project began in 1999 and rapidly attracted attention since then [29, 30]. The first generation of honeynet technology, also called GenI, is designed to control intruders and collect information with basic functionality. The second generation honeynet, GenII, is further implemented with the ability to capture encrypted communications of intruders at the kernel level.

Honeypots deployed in the open network help us to study and collect details about the behavior of intruders. However, honeypots are more suitable to be deployed in a dedicated network. Thus, some experts feel that honeypots should be classified as a special kind of IDS [7], which can help the system administrator detect attacks because it is a machine that no legitimate user is supposed to touch. Any probes toward the honeypot are considered suspicious and established connections to the honeypot obviously have malicious intent. In summary, a honeypot or honeynet is deployed as a passive monitoring component, and not as an active monitoring component that we are proposing in the Japonica framework. Further discussion of the Japonica framework is presented in Section 3.

More recent research explored the deployment of honeypots as shields of dedicated networks [41]. All traffic toward that network is first directed to honeypots. A decision is then made to either drop malicious connections or to allow legitimate connections to proceed. However, it may not be an ideal solution because honeypots are designed to attract attacks and lure intruders, but they are not meant to serve as a strong defense mechanism [30]. Deploying honeypots as the first line of defense may be dangerous with the lack of a strong defense. Besides, redirection is influenced by false positives and negatives, which increases the complexity of such a deployment.

Another relevant work is the open source project called Honeyd [26]. Honeyd is a network daemon that can control all unused IP addresses in a dedicated network. Low interaction is the disadvantage of this approach because it implements fake services. It can help detect attacks but these fake services may be easily discovered by intruders, due

to the low interaction capability. Therefore, in the Japonica framework, the honeypot is recommended to be implemented with real services. Due to the complete and early response provided in Japonica, the risk of a honeypot being compromised and becoming a stepping stone to attack other hosts is rapidly decreased, but it is still necessary to control outgoing traffic from honeypot.

2.2. Risk awareness

Traditionally, security technologies such as firewalls and IDSs perform their decision-making process in a static manner. For example, a firewall would decide whether a connection is legitimate or malicious, and would either allow it or block it accordingly. Likewise, an IDS would assess whether a traffic pattern is an intrusion or not. The result can consist of only two values: it is a threat, or it is not. There can be no "in betweens."

The early notions of risk awareness can be traced back to dynamic access control mechanisms. Thomas and Sandhu [35, 36, 37] proposed a concept called task-based access control. While traditional access control mechanisms employ a subject-object view, Thomas and Sandhu argued that this view is no longer sufficient for distributed and multi-tiered applications. Their approach performs access control at a higher level of abstraction in the form of long-lived tasks.

Another project named Seraphim [20] is a generic, theoretical framework for dynamic policy specification and enforcement. Seraphim differs from our work, where their work is highly generic while ours focuses on the network domain and actively uses honeypot technology as well.

Other work on dynamic access control include Knorr's work on Petri net workflows; Lin, Lee, and Chang's work on a dynamic access control mechanism in the area of information protection systems; and two more projects in the area of public key cryptography by Harn and Lin [12] and Yen and Laih [42].

One of the most recent works in this area is the approach by Teo, Ahn, and Zheng [34]. Their work focused on using dynamic and risk-aware network access management to dynamically control access to a network. They introduced the concepts of threat levels to represent the risk associated with a connection, and thresholds as boundaries that would invoke certain actions if bypassed by the threat levels. The Japonica framework uses similar concepts to enable early and complete response to unknown attacks.

3. The Japonica framework

In this section, we describe Japonica, a security framework that achieves early response through the dynamic orchestration of prevention, detection, and response mechanisms. The word Japonica is derived from the scientific name of the Japanese honeybee that we discussed in the introduction – *Apis cerana japonica*. We first discuss the objectives and requirements of the Japonica framework. We then present a detailed description of Japonica, including a model of the framework based on Colored Petri Nets (CPN). We also show how the framework fits into real world environments. We begin with a discussion of Japonica's three primary objectives:

Objective 1: Early and complete response to unknown attacks. The overall goal of Japonica is to actively detect and respond to unknown attacks as early as possible. It also aims to prevent future attacks of similar kind. Through this, an organization can recover from unknown attacks quickly and be protected from future attacks. The importance of this objective can be examined from the current trend of Internet attacks [1]. The SQL Slammer worm that spread around the Internet in just ten minutes could not be contained by manual, human intervention. Early response is critical [31] – however, at the same time, the response needs to be *complete* so that attacks would be contained properly. We reiterate that the attack should be addressed by preventative, detection, and response measures.

Objective 2: Accurate detection of attacks. The second objective of Japonica is to detect attacks more accurately. Current network IDSs still have problems distinguishing between legitimate traffic and malicious traffic, especially if they exhibit similar patterns [30]. Japonica aims to minimize incidents where legitimate traffic is mistaken to be malicious traffic. Therefore, it is required to minimize false positives as much as possible.

Objective 3: Scalability. In Objective 1, we mentioned that the response to an attack must be complete. This implies that the Japonica framework has to be deployed using more than one security technology, since a single technology by itself cannot ensure a complete response against attacks. Due to the diversity of security technologies, scalability is a crucial objective in the Japonica framework [2]. With the scalability of the Japonica framework, any intrusion detection system or firewall can be deployed within Japonica. To achieve this objective, a uniform message exchange format between different components has to be defined. Such a format would enable the different components to communicate with one another.

3.1. Requirements

Based on the objectives we just mentioned, we now describe the requirements of Japonica that need to be in place to support those objectives.

Requirement 1: Real-time coordination of prevention, detection, and response mechanisms. The first objective stresses early and complete response to unknown at-

tacks. In order to achieve this, all three aspects of security need to be considered: prevention, detection, and response. More specifically, these three mechanisms need to be coordinated in real time so that attacks can be responded to early. The framework would require input from distinct sources from the detection mechanisms, and invoke the prevention and response mechanisms accordingly.

Requirement 2: Generic mechanism to address a wide range of attacks. The first objective also stresses unknown attacks. To do so, the framework should not just cater for attacks that conform to a specific pattern [3]. It must be designed from the start to detect malicious patterns.

Requirement 3: Use of a mechanism that does not exhibit false positives. The second objective states that false positives should be minimized. Hence, Japonica would need to use a security mechanism that does *not* exhibit a huge number of false positives [30].

Requirement 4: Dynamic and risk-aware feedback-based approach to detect and respond to attacks. The ultimate goal of generating totally no false positives is ideal and highly desirable. However, in practice, any organization's network and environment can be very complex. Many variables come into play when we are trying to decide whether an event is an attack or not. Although Requirement 3 is very important, we need to be realistic about such variables. Otherwise, the framework would not be able to be deployed effectively in real world environments.

We attempt to address this issue by requiring the consideration of risk when interpreting the intent of an event. In other words, the framework needs to risk-aware. Having discussed this, we would immediately note that risk itself is not a static concept, but a dynamic one, especially in complex real world environments with many variables to consider. The dynamism of such environments needs to be a critical factor in the framework. To achieve dynamic risk assessment, one important technique that can be used is to make it feedback-based. Such dynamic and risk-aware mechanisms are already available [34].

Requirement 5: Uniform message exchange format. The third objective of the Japonica framework is scalability. As mentioned earlier, a uniform message exchange format needs to be designed to enable different components (including new components) to communicate with one another within the Japonica framework. The IETF's Intrusion Detection Message Exchange Format [10] (IDMEF) is a possible message exchange format to use; however, its implementation is not mature enough to be part of a Japonica implementation yet (although implementations certainly exist [15, 25, 40]). Instead of using IDMEF, we attempted to achieve the same functionality by modeling a message exchange format using Colored Petri Nets (CPN) for Japonica. This model is described in Section 3.3.

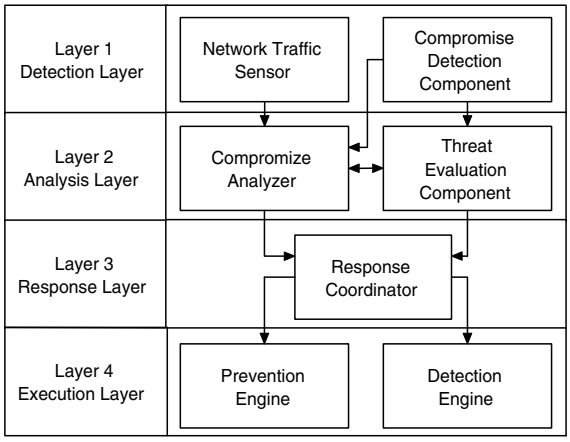


Figure 1. The Japonica framework stack

3.2. Japonica framework description

In this section, we describe the Japonica framework in detail. There are basically *seven* components in the Japonica framework: Compromise Detection Component (CDC), Network Traffic Sensor (NTS), Compromise Analyzer (CA), Threat Evaluation Component (TEC), Response Coordinator (RC), Prevention Engine (PE), and Detection Engine (DE). It is beneficial to organize the components of the framework as a layered stack, both for clarity and modularity, which would in turn facilitate implementation. The Japonica framework is illustrated as a layered stack in Figure 1.

Before we begin our discussion of the framework, we would like to make a note on the writing conventions for the rest of the paper. The Japonica components could be represented as acronyms, as described in the previous paragraph; however, using acronyms to describe seven different components could make the paper difficult to read. Therefore, we shall use their expanded forms to improve readability throughout this paper. The exception to this convention would be in certain sections where we discuss the Compromise Detection Component (CDC), a term that occurs fairly often (sometimes several times in one sentence). In such situations, we use the acronym CDC in place of the Compromise Detection Component.

3.2.1. Detection Layer (Layer 1). In Japonica, the main purpose of Layer 1, the Detection Layer, is to collect all kinds of information related to intrusions on the host. Therefore, the Compromise Detection Component (CDC) and the Network Traffic Sensor belong to Layer 1.

The CDC is a module that runs on the host. Its primary goal is to detect whether the host is compromised, and if so, what are the specific features of the compromise. Dif-

ferent CDCs may have different functions to detect various kinds of intrusions. For example, one CDC may detect whether there is any rootkit downloaded to the host, while another may detect unusual ports that are listening on the host. Other possible CDC functions include the detection of any accounts that are created without authorization on the host, or whether any critical files have been replaced by an attacker. From this, we can see that a CDC by itself can perform either a simple lightweight function, or a complex heavyweight function. However, when we run many CDCs with different functions together on a single host, the situation becomes increasingly complex. Due to the complexity, it is recommended that organizations adjust their additional CDCs according to the requirements of specific environments. For example, in the environment of a financial institution, a CDC can be designed to detect stolen credit card numbers.

The Network Traffic Sensor is a component that can be easily understood. It keeps the records of suspicious network traffic, including source port, destination port, source IP, destination IP, protocol type and packet size.

3.2.2. Analysis Layer (Layer 2). The Analysis Layer accepts important information from Layer 1. The primary goal of Layer 2, the Analysis Layer, is to analyze and determine whether the input is a malicious activity or not based on the threat level.

The concept of a threat level is very crucial in the Japonica framework. The threat level determines the risk associated with an entity. The more suspicious the entity behaves, the higher the threat level will be. The threat level is meant to be dynamic – it will change dynamically based on the risk that is perceived in the environment. This is how it captures the dynamism of the environment. In the context of this research, the threat level is a quantifiable measure of the risk associated with a set of one or more CDCs. Each unique set of possible CDCs will have its own threat level. If we have n CDCs, the number of possible threat levels would be $2^n - 1$.

The next concept we need to consider is the threshold. The threshold represents the tolerance towards suspicious events. Unlike the threat level which is dynamic, the threshold is statically defined. Once the threat level exceeds the threshold, further actions will be taken in Layer 3.

In other words, the Compromise Analyzer communicates with the CDC, Network Traffic Sensor and Threat Evaluation Component to dynamically determine whether there is an attack going on or not. If so, the Compromise Analyzer will send the message with the necessary information to the Response Coordinator. The Threat Evaluation Component is the component that determines whether the current threat level has exceeded the threshold according to input from CDCs and pre-defined threat level policies.

For example, when a host is under attack from the CodeRed II worm, a file named "root.exe" will be added to the host [11]. At this point, one of the CDCs on that host that is tracking file integrity will send out a message to the Compromise Analyzer. The threat level of the downloaded file will be set, say 5, according to the threat level policy defined by the administrator [34]. After the host is infected, CodeRed II will attempt to connect to other hosts using generated random IP addresses. The CDC will track all outgoing network traffic, and then send a message to Compromise Analyzer. At the moment, the current threat level will increase; say from 5 to 12 according to threat level policy. After this stage, the Compromise Analyzer would inquire the Threat Evaluation Component to find out the current threat level policy. Say the administrator defined the threshold to be 10. Since the threshold has been exceeded, the Compromise Analyzer sends the file information and outgoing traffic information to the Response Coordinator. The Response Coordinator is located in the Response Layer, which we shall discuss in the next section.

3.2.3. Response Layer (Layer 3). Layer 3 is the Response Layer, which consists of only the Response Coordinator. Layer 3 of Japonica is an intermediary layer, and its main purpose is to generate response rules for Layer 4. According to the analysis results and information input from Layer 2, the Response Coordinator generates corresponding prevention rules and detection rules and sends them to the Prevention Engine and Detection Engine on hosts under attack and also notify system administrators. Consider the previous CodeRed II example. The Response Coordinator generates the detection rules according to the file information and outgoing traffic information from the Compromise Analyzer and sends the detection rules to the Detection Engine. The Response Coordinator also generates prevention rules and sends them to the Prevention Engine.

3.2.4. Execution Layer (Layer 4). The Prevention Engine and Detection Engine are located in Layer 4, the Execution Layer. The main feature of the Prevention Engine is to filter out and block malicious threats, and the Detection Engine is responsible for detecting any ongoing attacks based on the detection rules generated by the Response Coordinator. The difference between Layer 1 and the Detection Engine is that Layer 1 is mainly host-based while the Detection Engine is network-based and relies on pattern matching functionality. In other words, the Prevention Engine and Detection Engine are still using static threat detection and prevention rules. The dynamic rule adjustments are processed in Layer 2, and the unknown attack detection functionality is carried out in Layer 1.

Recalling our CodeRed II example, after the Detection Engine receives detection rules from the Response Coor-

$CPN = (\Sigma, P, T, A, N, C, G, E, I)$, where:

Σ	is a finite set of non-empty types, also called color sets.
P	is a finite set of places.
T	is a finite set of transitions.
A	is a finite set of arcs, where $P \cap T = P \cap A = T \cap A = \emptyset$
N	is a node function, where $N : A \rightarrow P \times T \cup T \times P$
C	is a color function, $C : P \rightarrow \Sigma$
G	is a guard function, $G : T \rightarrow expressions$, such that $\forall t \in T : [Type(G(t)) = B \wedge Type(Var(G(t))) \subseteq \Sigma]$
E	is an arc expression function, $E : A \rightarrow expressions$, such that $\forall a \in A : [Type(E(a)) = C(p)MS \wedge Type(Var(E(a))) \subseteq \Sigma]$, where p is the place of $N(a)$.
I	is an initialization function, $I : P \rightarrow closedexpressions$, and $\forall p \in P : [Type(I(p)) = C(p)MS]$

Table 1. Formal definition of Colored Petri Nets (CPN)

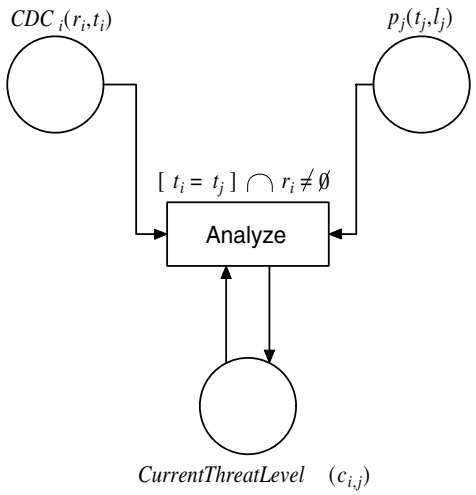


Figure 2. Feedback-based threat level evaluation represented using Colored Petri Nets

dinator, it will reset the engine to apply the new detection rules. The Prevention Engine will also apply new prevention rules. Finally, the Prevention Engine blocks any traffic with the same signature thus stopping the spread of CodeRed II.

3.3. CPN-based Japonica description

We attempt to model the Japonica framework using Colored Petri Nets (CPN) which is well-known as a powerful graph theory in the system modeling area [19]. Jensen defined CPN as an advanced Petri net theory in 1992 [14]. Petri nets include three basic components: places, transitions and arcs [24]. The difference between CPN and Petri nets is the idea of color sets in CPN, which can be viewed as an abstract data type in a programming language. The formal definitions of CPN are given in Table 1.

There are several advantages to model the Japonica framework with CPN. First of all, a uniform message exchange format is well-defined when color sets are used. For instance, the traffic record transferred from the Network Traffic Sensor to the Compromise Analyzer is initially declared with the traffic color set, which is (src, dst, src_port, dst_port, protocol, packetsize, hashvalue). After the traffic color set is set, all traffic records must follow the same declaration, regardless of whether they originate from the Network Traffic Sensor to the Compromise Analyzer, or from the Compromise Analyzer to the Response Coordinator. The remaining color set declaration of Japonica is shown in Appendix A. With the uniform message exchange format, the Response Coordinator can coordinate with various Prevention Engines and Detection Engines, thus achieving early and complete response. Another advantage is the flexibility of representing feedback-based threat level concepts and the response mechanism in the Japonica framework. With the graphic representation of CPN, it is especially useful to detect and solve the problem of dynamic structures. Another key advantange is the convenience of Design/CPN [39], the standard CPN design tool, which allows us to clearly and graphically demonstrate how the components interact among themselves in the Japonica framework.

3.3.1. Threat level mechanism. Based on the CPN theory previously explained, Figure 2 illustrates how we can model feedback-based threat level evaluation in the Japonica framework. We use the following notation to describe the elements and functions in Figure 2:

Let n be the number of CDCs. $Place = (CDC_1, CDC_2, \cdots, CDC_n, P, C)$ is a finite set of places. Each $CDC_i = (r_i, t_i)$ where t_i is the type of CDC and r_i is the corresponding compromise result. The threat policy is $P = \{p_1, p_2, \cdots, p_n\}$. Each $p_j = (t_j, l_j)$, where t_j is the type of CDC and l_j is the corresponding weight of the threat level. The current threat level $C = c_{i,j}, 0 < c_{i,j} < max$, where max is the user-defined maximum threat level allowed, and $c_{i,j}$ is the threat level of CDC_i according to threat level policy p_j. Lastly, $Transition = (Analyze)$ is the finite set of transitions.

The return value of the guard function is Boolean. The guard function G in Figure 2 is $[t_i = t_j] \cap r_i \neq \emptyset$. This

means that the function will return true only if the type of CDC input matches the policy type and a corresponding compromise result exists.

Function E within the $Analyze$ transition is the threat level evaluation function, where $c_{i,j} = E(CDC_i, p_j)$. Therefore, the total threat level is:

$$\sum_{i=1}^{n} c_{i,j} = \sum_{i=1}^{n} E(CDC_i, p_i) \quad (1)$$

As an example, in Figure 2, the CDC place could send out a message such as ('c:\windows\root.exe', 'file') during a CodeRed II attack. The second part of the message indicates the CDC type, while the first part carries the location of the file added on host.

The threat level place p stores the information about the policies of the corresponding threat level. The $Analyze$ transition can be mapped to the Compromise Analyzer component, which was described in Section 3.2.2. It determines the current threat level according to the threat level policies and input from various CDCs. In Figure 3, we show that if the current threat level exceeds the threshold, the $Respond$ transition will generate detection rules and prevention rules.

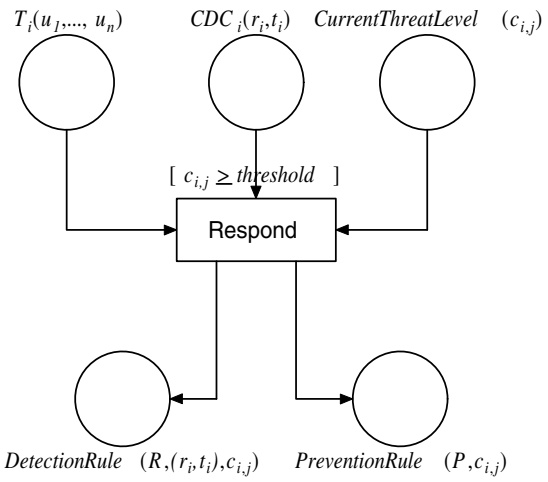


Figure 3. Response Coordinator represented using Colored Petri Nets

3.3.2. Response mechanism. Figure 3 illustrates how we modeled the response mechanism in Japonica. We use the following notation to describe the elements and functions in Figure 3:

$Place = \{T_i, CDC_i, CurrentThreatLevel, DetectionRule, PreventionRule\}$ is a finite set of places. Each ith network traffic record is defined to be $T_i = (u_1, u_2, \cdots, u_j, \cdots, u_k, \cdots, u_n)$, where u_j and u_k are different elements of network traffic records. We define Detection Rules = $(R, c_{i,j}, r_i, l_i)$, where $R \subseteq T_i$. We define Prevention Rules = $(P, c_{i,j})$, where $P \subseteq T_i$. $Transition = (Respond)$ is the finite set of transitions.

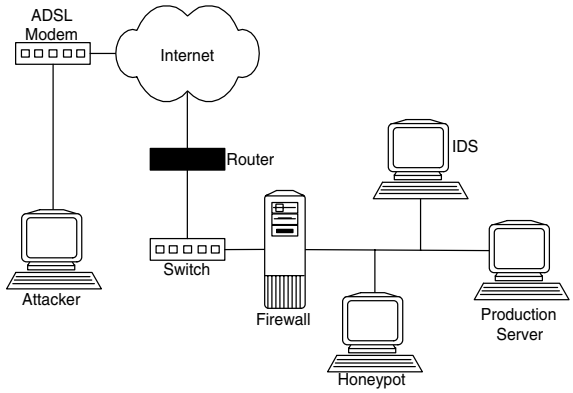


Figure 4. Japonica in a network environment

The return value of the guard function is Boolean. The guard function G in Figure 3 is $c_{i,j} \geq threshold$, where $threshold$ is the user-defined threat level that is allowed in current situation.

Function N within the $Respond$ transition is the traffic elements pre-selection function for generating detection rules and prevention rules, where $R = N(T_i, t_i)$ and $P = N(T_i, t_i)$, and t_i is the type of CDC_i.

Function K within the $Respond$ transition is the function that generates detection rules, where $K(R, (r_i, t_i), c_{i,j})$ produces detection rules. Likewise, function M within the $Respond$ transition is the function that generates prevention rules, where $M(P, c_{i,j})$ produces prevention rules.

In Figure 3, we show that the $Respond$ transition will only be triggered when the current threat level exceeds the threshold. The $Respond$ transition is the Response Coordinator in Japonica, which is used to generate detection rules and prevention rules. Through this mechanism, different CDCs can result in the generation of different detection rules and prevention rules because of function N, which is determined by the CDC type. This design also increases the flexibility and scalability of Japonica. If more CDCs are to be deployed at a later stage, only function N needs to be revised.

3.4. Real-world environment

In this section, we discuss how the Japonica framework fits into a typical network environment. Figure 4 is a common real-world network diagram, which includes a firewall, a honeypot, a production server and an IDS. The configuration of the production server and the honeypot are the same.

The only difference is that there is no sensitive data stored on the honeypot. In other words, the honeypot provides real services with no real data.

In this environment, there are four different types of CDCs implemented on the honeypot: a port-monitoring CDC, an account-monitoring CDC, an outgoing-traffic-monitoring sensor and a file-integrity-checking CDC. The port-monitoring CDC sends a message to the Analyzer when there is any new port opened. Based on attacks on the machines we discovered and other work on backdoor detection [43], a new open port is a crucial feature on a compromised machine. The account-monitoring CDC sends a message out when there is any additional account created on the honeypot. The illegal privilege increase for normal user accounts is another important feature signaling a compromise [9]. Any account created on a honeypot is suspicious because there is not supposed to be any legitimate user account that would be added to the honeypot. The file-integrity-checking CDC is responsible for tracking any files that are created or downloaded [4] to the honeypot since there is no legitimate reason for such activities on a honeypot.

The outgoing-traffic-monitoring sensor records any outgoing traffic from the honeypot, since there should not be any legitimate outgoing traffic from the honeypot. This way, the outgoing-traffic-monitoring sensor matches the definition of the Network Traffic Sensor, which keeps records of suspicious outgoing network traffic only. Besides, the outgoing network connection is an identical feature for hosts that are infected by Internet worms [38].

Instead of pattern matching, we try to model the intrusion behaviors with various kinds of CDCs and determine risk by dynamic threat level evaluation. This satisfies Requirement 2 in Section 3.1, which states that we need a generic mechanism to address a wide range of attacks. Besides, since there is no sensitive data stored on the honeypot, the risk allowed is higher than that of the production server.

The advantage of implementing CDCs on the honeypot is to address Requirement 3 described in Section 3.1, which mandates the use of a mechanism that does not exhibit false positives. Due to the nature of the honeypot, there is no legitimate activity on it. It is therefore much easier to determine the threat level of such suspicious activities, thus decreasing false positives dramatically.

The Network Traffic Sensor is also implemented on the honeypot, which records all outgoing traffic from the honeypot. The Compromise Analyzer and Response Coordinator are also implemented on the honeypot. In such a deployment, a potential issue may be that an attacker who manages to break into the honeypot would be able to modify or compromise the CDC, Network Traffic Sensor, Compromise Analyzer, and Response Coordinator. To mitigate this, these components would have to be installed in a manner that is unmodifiable by any unauthorized user on the honeypot. For example, these components could be run from a physical read-only medium such as a CD-ROM, or installed and executed at the kernel level such that even a superuser would be not able to modify them.

In Figure 4, the firewall is deployed as the Prevention Engine because of its traffic filtering feature, while the IDS is deployed as the Detection Engine. The main reason that the Compromise Analyzer and Response Coordinator are implemented on a honeypot instead of on the same host of the Detection Engine or Prevention Engine is due to overhead issues. If the Compromise Analyzer and Response Coordinator are deployed on the Detection Engine or Prevention Engine, they might have difficulty in accomplishing real-time detection or prevention tasks.

4. Proof-of-concept experiment

In order to examine the feasibility of the Japonica framework, we carried out an experiment based on a case study on SQL Slammer.

4.1. Fulfilling the requirements

Before we commence the discussion of our proof-of-concept experiment, it is important to examine the criteria in which the experiment should be tested, and how the case study is selected.

The criteria we chose were used to validate most of the requirements defined in Section 3.1. Since we are primarily concentrating on demonstrating the feasibility of our approach rather than building a full-fledged implementation, we decided to perform the experiment to validate the most important requirements – in this case, the first four requirements. The reasons and descriptions of the criteria for each requirement are given as follows:

Criterion to fulfill Requirement 1: Requirement 1 states that the framework implementation must coordinate prevention, detection, and response mechanisms in real time. Validating this requirement is important, since it would fulfill the cornerstone objective of the Japonica framework, which is early and complete response.

Criterion to fulfill Requirement 2: Requirement 2 states that the mechanism must be generic enough to address a wide range of attacks, including unknown ones. This is a very important requirement since it would fulfill the first objective as well. To scale down the fulfillment of this requirement, we chose to use SQL Slammer as the attack in the experiment. SQL Slammer was chosen because it was relatively new and had the characteristics of a previously unknown attack. Therefore, it is suitable as the subject of a case study to assess the ability of a Japonica implementa-

tion to detect unknown attacks. We are currently devising more experiments to handle more attacks in the future.

Criterion to fulfill Requirement 3: Requirement 3 states the need to use a mechanism that does not show false positives. As described in Section 3.4, one of the components in a real world deployment of Japonica is a honeypot – the honeypot in this case is the mechanism that we would use to fulfill Requirement 3. The reason behind this decision is that a honeypot does not have any use for legitimate users – its primary aim is to allow attackers to abuse it so that the attackers patterns can be studied. Therefore, all traffic going in and out of the honeypot can be classified as malicious traffic, thus fulfilling our requirement of a mechanism that does not exhibit false positives. Note that although we said all traffic can be classified as malicious, there are a few exceptions. One such exception would be broadcast protocols. We shall take the Address Resolution Protocol (ARP) as an example. In order for a new host to find out the MAC address of a given IP address, the ARP protocol would broadcast an ARP request message to the network. If such traffic reaches the honeypot, it would look like incoming traffic. However, it would not be wise to classify this as malicious since it is merely a legitimate ARP request. Therefore, the threat level policy for outgoing traffic should be defined to be not as sensitive as other more suspicious features indicating compromise, such as a downloaded file on the honeypot. In this way, once an automatic attack occurs in the honeypot, such as an Internet worm, the threat level will exceed the threshold in a short time because it generates a huge amount of outgoing connection attempts.

Criterion to fulfill Requirement 4: Requirement 4 stated the need to use a dynamic and risk-aware feedback-based approach to detect and respond to attacks. This requirement is very important since the threat level concept is crucial in Japonica's framework. Without the dynamic feedback-based threat level determination mechanism, Japonica will be no different compared to other frameworks that use static pattern matching functionality.

Requirement 5: Requirement 5 stated that Japonica should work using a uniform message exchange format. While this is desired, it is not very critical when it comes to demonstrating a proof-of-concept implementation. Furthermore, the message exchange format needs to be designed carefully. Due to time and resource limitations, designing and developing the message exchange format was deferred as future work.

4.2. Experimental environment

Since we chose to deal with a highly virulent Internet worm that could potentially be let out into the wild, we were very careful when we set up the environment for our experiment. To ensure that the worm does not go loose, it needs to be isolated. At the same time, the limited environment needs to be able to demonstrate the fulfillment of the requirements we outlined above.

To deploy such an environment, we set up a virtual, isolated network using virtual machines running on VMware. A diagram of the environment is shown in Figure 5. It can be viewed as a simulated subset environment of Figure 4.

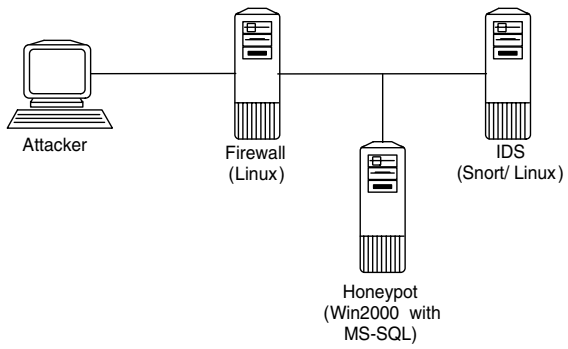


Figure 5. Virtual, isolated environment for our experiments

The components were set up as follows. The Prevention Engine was implemented as a firewall running on the Linux operating system. The Detection Engine was implemented using Snort, a popular open source IDS, which was also run on the Linux platform. The firewall and IDS were chosen as the Prevention Engine and Detection Engine respectively due to straightforward reasons: the firewall blocks malicious traffic, hence it prevents attacks from being successfully carried out; the IDS detects attacks, hence its suitability as a Detection Engine.

Apart from the Prevention Engine, the firewall also runs the Network Traffic Sensor, Compromise Analyzer, Threat Evaluation Component, and the Response Coordinator. The firewall was selected to run these components because of its location. It is the first point of entry into the network, and therefore it would be well-positioned to capture all traffic and analyze them accordingly before generating a response.

The honeypot was deployed using Windows 2000 with a vulnerable version of SQL Server. SQL Server was intentionally left vulnerable to enable the SQL Slammer worm to compromise it. The honeypot ran one single CDC process, where its sole purpose was to capture outgoing traffic and send it to the Compromise Analyzer running on the firewall.

4.3. Experiment and results

Before we describe how the experiment was conducted, it would be beneficial to highlight a few technical characteristics of the SQL Slammer worm.

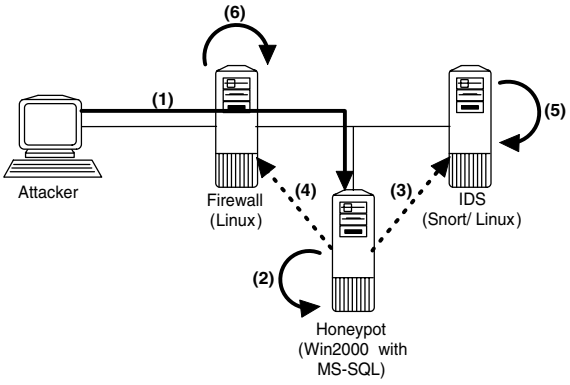
The SQL Slammer worm is an extremely high-speed worm (it was reported to have performed at a rate of 55 million scans per second after three minutes [17]). A typical SQL Slammer scan involves the transmission of a UDP packet with a specially crafted 376-byte data payload to a random destination IP address on port 1434 (one of the ports that SQL Server listens on). The payload would compromise the SQL Server process at the target IP address via a buffer overflow attack in its data payload. An SQL Slammer attack typically sends this UDP packet to random IP addresses generated at high speed. Once another host is compromised, it too would send the same packet to generated random IP addresses. When many hosts are compromised this way, network resources are used up heavily, thus slowing down the network.

We will now present our experiment. The experiment was carried out by specially directing the SQL Slammer worm from the attacker's machine to the honeypot. In the experiment, the honeypot was successfully compromised since it was running an unpatched SQL Server. The honeypot then started sending the crafted UDP packets to randomly-generated IP addresses.

When this happened, our CDC detected the outgoing traffic and forwarded it to the Compromise Analyzer on the firewall. The Compromise Analyzer would identify that the attack consisted of many packets with the same characteristics: the UDP protocol, port 1434, and the 376-byte data payload size. As a result of the frequency of the attack, the Threat Evaluation Component would increase the threat level of this attack quickly. In the experiment, we did not set the Threat Evaluation Component to be very sensitive to attacks because outgoing traffic may include broadcast traffic as we mentioned in Section 4.1.

However, due to the nature of SQL Slammer, once the honeypot was infected, thousands of outgoing packets were sent out to infect other hosts, thus resulting in the threat level rapidly increasing over the threshold. The Compromise Analyzer would send these characteristics and the threat level to the Response Coordinator. The Response Coordinator then generated the firewall rule and IDS signature to counter and detect the attack respectively. The firewall and IDS then automatically reloaded themselves to reflect the new rules. Once this was done, SQL Slammer was successfully stopped from propagating out of the network, and the IDS would generate alerts based on the new signature. Figure 6 summarizes the steps in our experiments.

The duration of our experiment was just 28 seconds. In that short duration, SQL Slammer generated 217,483 packets at a rate of approximately 4,400 scans per second. We show the generated Snort signature and the Snort alert in Figures 7 and 8 respectively. The destination IP address in Figure 7 was intentionally masked for the purposes of this paper.



(1) SQL Slammer launched.
(2) Honeypot is infected:
- CDC detects outgoing traffic.
- CA inquires TEC and determines it has exceeded threshold.
- RC generates DE rules and PE rules.
(3) RC sends DE rules to IDS.
(4) RC sends PE rules to firewall.
(5) IDS restarts and applies new signatures.
(6) Firewall restarts and applies new rules.

Figure 6. Detecting and responding to the SQL Slammer attack using Japonica

```
# Tue Apr 29 14:59:26 EDT 2003
alert udp $EXTERNAL_NET any -> any 1434
(msg:"unknown worm propagation attempt";
dsize:376; classtype:misc-attack;
sid:1000001; rev:1;)
```

Figure 7. Generated Snort signature

4.4. Rule generation algorithm

We conclude this section with a brief discussion of the rule generation algorithm (the algorithm that generates Snort signatures). At this point, we are currently refining the rule generation algorithm to produce signatures that would capture a wider range of attacks. A lot of challenges still remain to integrate the rule generation algorithm so that it will work with different honeypots, firewalls, and intrusion detection systems. To be effective, the algorithm needs to intelligently examine various properties of network packets for anomalies.

In our experiment, the Response Coordinator used the rule generation algorithm in a cohesive manner within our experimental environment. In other environments, the algorithm would have to work with other forms of data. For example, in the financial services sector, the algorithm and Japonica components would have to process context-specific data such as credit card information and account information. This is clearly not a trivial exercise. We also

```
[**] [1:1000001:1] unknown worm
propagation attempt [**]
[Classification: Misc Attack] [Priority:
2]
04/29-14:59:26.384613 172.16.21.130:2081
-> xxx.xxx.xxx.xxx:1434
UDP TTL:128 TOS:0x0 ID:33421 IpLen:20
DgmLen:404 Len: 376
```

Figure 8. Snort alert generated based on the new signature

need to address other factors, such as interoperability with non-security systems and deployment of a Japonica implementation on different platforms.

5. Discussion and future work

So far, we have demonstrated the ability of Japonica to detect unknown attacks such as SQL Slammer. In order to smoothly deploy the Japonica framework, however, a few issues still need to be resolved. These issues are discussed in this section.

First of all, the possible false negatives are always a crucial problem for honeypot-centric defense mechanisms [30]. As Spitzner mentioned [30], it is still possible that skillful individual attackers may directly attack production servers instead of the honeypot, and we believe this to be the main reason why other honeypot-related defense mechanisms deploy honeypots as a front-line shield [41]. This would force all incoming traffic to go through the honeypot first. Therefore, we do not suggest honeypots as a replacement of current network defense mechanisms; rather, we deploy honeypot as a complementary defense mechanism. In the Japonica framework, a distributed IDS [28] or distributed firewall [5] can be deployed as another CDC as well, which can rapidly decrease the possible false negatives. Furthermore, in the case of automated attacks such as Internet worms, the chances for the production server and honeypot to be attacked are equal.

In future experiments and Japonica implementations, we would have to ensure that a honeypot that is infected by a worm would not help to propagate the worm inside the internal network. Also, since the firewall rule is generated after the worm has infected the honeypot, there is a possibility that the honeypot may send outgoing worm traffic onto the Internet before the rule is enabled on the firewall. This latency issue is not unique to Japonica; it is present in current intrusion prevention systems as well [16, 23]. To address these issues, data control techniques would have to be used as a safeguard between the honeypot and the rest of the network. One practical implementation would be to use a Layer 2 filtering bridge that could be deployed between the honeypot and the rest of the network. Having no IP address, the bridge would be transparent and "invisible" to attackers, but it would still be able to filter and block traffic. We could use such a mechanism to allow malicious traffic to enter the honeypot but only a restricted set of traffic to exit it. Various potential schemes could be used, such as the ability to allow only traffic packets that conform to a specific criteria to be returned. Clearly, Japonica control traffic would have to be allowed to leave the honeypot. By implementing such data control mechanisms, the likelihood of an infected honeypot helping to spread a worm or any other form of self-propagating code would be significantly reduced.

Another concern about the Japonica framework is the difficulty of transforming unstructured forensic data on the honeypots into structured data for the purpose of generating prevention rules and detection rules. We have designed four different kinds of CDCs for this purpose; however, it still requires a lot of effort to further refine the design of the CDCs in order to detect unknown attacks based on behaviors instead of pattern matching. Most of the work related to this direction is still ad-hoc and no complete survey has yet to emerge. However, due to the scalability of Japonica, it will be straightforward to add another different CDC into the framework. This would make any design enhancements easier and more flexible.

Communications between each component is also an issue. In order to achieve an early and complete defense, a secure communication channel and protocol among components must be developed. The traffic generated by the protocol would also have to be masked from attackers.

Despite the above issues about Japonica, it is obvious that honeypot-centric defense mechanisms can dramatically reduce the analysis load due to the huge amount of log files on the production server. Besides, the cooperation among different network areas in which the Japonica framework is deployed would help to facilitate the task of improving the risk-aware response mechanism. For example, say departments A and B both deploy the Japonica framework. The CDCs of department A could provide input to the Threat Evaluation Component in department B, only with smaller weights. In this way, a distributed Japonica framework can be implemented and early response can be achieved more thoroughly.

Containing worms and other future unknown attacks is a very difficult endeavor, and many technological challenges still remain [18]. We are working to address such issues within the Japonica framework. Our current strategy is to build a complete Japonica framework implementation and carry out more experiments involving unknown attacks.

6. Conclusion

We have proposed a security framework called Japonica to detect and respond to unknown attacks at the early stage through the dynamic orchestration of prevention, detection, and response mechanisms. We identified important requirements to support the proposed framework and corresponding system entities. Also, we described our model using Colored Petri Nets to discover a uniform message exchange format among the entities. Our results indicated that Japonica could successfully prevent the spread of SQL Slammer without human intervention. We believe that our framework can counter other forms of sophisticated Internet attacks and we are currently investigating how we can extend our approach to support such attacks.

Acknowledgments

This work was partially supported at the Laboratory of Information of Integration, Security and Privacy at the University of North Carolina at Charlotte by the grants from National Science Foundation (NSF-IIS-0242393) and Department of Energy Early Career Principal Investigator Award (DE-FG02-03ER25565).

References

[1] Overview of attack trends. Technical report, CERT Coordination Center.

[2] J. Allen, A. Christie, W. Fithen, J. McHugh, J. Pickel, and E. Stoner. State of the practice of intrusion detection technologies. Technical Report CMU/SEI-99-TR-028, CMU/SEI, January 2000.

[3] O. Arkin. Trace-back: A concept for tracking and profi ling malicious computer attackers. Technical report, @Stake, 2003.

[4] AusCERT. Know thy attacker, July 2002. http://www.auscert.org.au/render.html?it=2000&cid=1920.

[5] S. M. Bellovin. Distributed fi rewalls, November 1999. http://www.research.att.com/ smb/papers/distfw.html.

[6] B. Cheswick. An evening with Berferd in which a cracker is lured, endured, and studied. In *Proceedings of the USENIX Winter 92 Conference*, January 1992.

[7] W. R. Cheswick, S. M. Bellovin, and A. D. Rubin. *Firewalls and Internet Security: Repelling the Wily Hacker*. Addison-Wesley, 2nd edition, February 2003.

[8] F. Cohen. The deception toolkit. http://www.all.net/dtk/.

[9] R. K. Cunningham and C. S. Stevenson. Accurately detecting source code of attacks that increase privilege. In *Proceedings of the 4th International Workshop on Recent Advances in Intrusion Detection (RAID 2001)*, pages 104–116, 2001.

[10] D. A. Curry and H. Debar. Intrusion detection message exchange format data model and extensible markup language (XML) document type defi nition. IETF Intrusion Detection Working Group, November 2002.

[11] eEye Digital Security. CodeRedII worm analysis. Advisory AL20010804, August 2001. http://www.eeye.com/html/Research/Advisories/ AL20010804.html.

[12] L. Harn and H. Lin. Integration of user authentication and access control. In *IEE Proceedings-E*, volume 139, number 2, pages 139–143, 1992.

[13] B. Heinrich. *The Thermal Warriors: Strategies of Insect Survival*. Harvard University Press, April 1999.

[14] K. Jensen. Coloured petri nets. basic concepts, analysis methods and practical use. *Monographs in Theoretical Computer Science*, 1, 1997.

[15] J. McAlerney and A. Migus. Libidmef. http://sourceforge.net/projects/libidmef/.

[16] E. Messmer. Intrusion prevention systems raise hopes, concerns. *Computerworld*, November 2002. http://www.computerworld.com/securitytopics/security/ story/0,10801,75630,00.html.

[17] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver. The spread of the Sapphire/Slammer worm. Technical report, January 2003. http://www.caida.org/outreach/papers/2003/sapphire/ sapphire.html.

[18] D. Moore, C. Shannon, G. M. Voelker, and S. Savage. Internet quarantine: Requirements for containing self-propagating code. In *Proceedings of the 2003 IEEE Infocom Conference*, San Francisco, CA, April 2003.

[19] T. Murata. Petri nets: Properties, analysis and applications. In *Proceedings of IEEE*, volume 77, pages 541–580, April 1989.

[20] P. Naldurg and R. H. Campbell. Dynamic access control policies in Seraphim. Technical Report UIUCDCS-R-2002-2260, Computer Science Department, University of Illinois at Urbana-Champaign, February 2002.

[21] M. Ono, T. Igarashi, E. Ohno, and M. Sasaki. Unusual thermal defence by a honeybee against mass attack by hornets. *Nature*, (377):334–336, 1995.

[22] M. Ono, I. Okada, and M. Sasaki. Heat production by balling in the Japanese honeybee, Apis cerana japonica as a defensive behavior against the hornet, Vespa simillima xanthoptera (Hymenoptera: Vespidae). *Experientia*, (43):1031–1032, 1987.

[23] J. Pescatore. Enterprise security moves toward intrusion prevention. *CSO Online*, September 2003. http://www.csoonline.com/analyst/report1771.html.

[24] J. L. Peterson. *Petri Net Theory and The Modeling of Systems*. Prentice Hall, 1981.

[25] S. Poppi. Threatman. http://sourceforge.net/projects/ threatman/.

[26] N. Provos. Honeyd. http://www.citi.umich.edu/u/provos/ honeyd/.

[27] B. Schneier. Crypto-gram newsletter, November 2002. http://www.schneier.com/crypto-gram-0211.html.

[28] S. R. Snapp, J. Brentano, G. V. Dias, T. L. Goan, L. T. Heberlein, C.-L. Ho, K. N. Levitt, B. Mukherjee, S. E. Smaha, T. Grance, D. M. Teal, and D. Mansur. DIDS (Distributed Intrusion Detection System) - motivation, architecture, and an early prototype. In *Proceedings of the 14th National Computer Security Conference*, pages 167–176, October 1991.

[29] L. Spitzner. *Honeypots: Tracking Hackers*. Addison-Wesley, September 2002.

[30] L. Spitzner. The honeynet project: Trapping the hackers. *IEEE Security and Privacy Magazine*, March/April 2003.

[31] S. Staniford, V. Paxson, and N. Weaver. How to 0wn the Internet in your spare time. In *Proceedings of the 11th USENIX Security Symposium (Security '02)*, San Francisco, CA, August 2002.

[32] C. Stoll. Stalking the wily hacker. *Communications of the ACM*, May 1988.

[33] C. Stoll. *The Cuckoo's Egg: Tracking a Spy through the Maze of Computer Espionage*. Pocket Books, 1989.

[34] L. Teo, G.-J. Ahn, and Y. Zheng. Dynamic and risk-aware network access management. In *Proceedings of the 8th ACM Symposium on Access Control Models and Technologies (SACMAT 2003)*, pages 217–230, Como, Italy, June 2003.

[35] R. K. Thomas and R. S. Sandhu. Towards a task-based paradigm for flexible and adaptable access control in distributed applications. In *Proceedings on the 1992-1993 Workshop on New Security Paradigms*, pages 138–142, Little Compton, RI, 1993.

[36] R. K. Thomas and R. S. Sandhu. Conceptual foundations for a model of task-based authorizations. In *Proceedings of the 7th IEEE Computer Security Foundations Workshop*, pages 66–79, Franconia, NH, June 1994.

[37] R. K. Thomas and R. S. Sandhu. Task-based authorization controls (TBAC): A family of models for active and enterprise-oriented authorization management. In *Proceedings of the IFIP WG11.3 Workshop on Database Security*, Lake Tahoe, CA, August 1997.

[38] T. Toth and C. Kruegel. Connection-history based anomaly detection. In *Proceedings of the 2002 IEEE Workshop on Information Assurance*, United States Military Academy, West Point, NY, June 2002.

[39] University of Aarhus. Design/CPN. http://www.daimi.au.dk/designCPN/.

[40] Y. Vandoorselaere. Prelude hybrid IDS. http://www.prelude-ids.org/.

[41] N. Weiler. Honeypots for distributed denial of service attacks. In *Proceedings of the 11th IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'02)*, June 2002.

[42] S.-M. Yen and C.-S. Laih. On the design of dynamic access control scheme with user authentication. *International Journal of Computers and Mathematics with Applications*, 25(7):27–32, 1993.

[43] Y. Zhang and V. Paxson. Detecting backdoors. In *Proceedings of the 9th USENIX Security Symposium*, Denver, CO, August 2000.

Appendix A

The CPN color set for the Japonica framework is defined as follows. A discussion of the traffic color set and the Japonica CPN model was presented in Section 3.3.

```
color realrange = int with 1..100;
color threatlevel = realrange;
color result = string;
color resulttype = string;
color CDC = product resulttype * result;
color payloadhash = string;
color src = string;
color dst = string;
color srcport = int;
color dstport = int;
color protocoltype = with tcp|udp|icmp;
color packetsize = int;
color traffic = product src * dst * srcport
                * dstport * protocoltype *
                packetsize * payloadhash;
color opr = string;
color weight = realrange;
color threshold = int;
color threatpolicy = product resulttype *
                     opr * weight;
color msg = string;
color resultset = product result *
                  resulttype;
color preventionrule = product traffic *
                       msg;
color detectionrule = product traffic *
                      msg * threatlevel;
```