EMBEDDED MONITORS FOR DETECTING AND PREVENTING INTRUSIONS IN CRYPTOGRAPHIC AND APPLICATION PROTOCOLS

Sachin P. Joglekar, B.E.

Thesis Prepared for the Degree of

MASTER OF SCIENCE

UNIVERSITY OF NORTH TEXAS

December 2003

APPROVED:

Stephen R. Tate, Major Professor
Armin R. Mikler, Committee Member
Ram Dantu, Committee Member
Oscar N. Garcia, Dean of the College of Engineering
Sandra L. Terrell, Interim Dean of the Robert B. Toulouse School of Graduate Studies

Joglekar, Sachin P., *Embedded monitors for detecting and preventing intrusions in cryptographic and application protocols*, Master of Science, (Computer Science), December 2003, 46 pp., 3 tables, 7 illustrations, references, 42 titles.

There are two main approaches for intrusion detection: signature-based and anomaly-based. Signature-based detection employs pattern matching to match attack signatures with observed data making it ideal for detecting known attacks. However, it cannot detect unknown attacks for which there is no signature available. Anomaly-based detection builds a profile of normal system behavior to detect known and unknown attacks as behavioral deviations. However, it has a drawback of a high false alarm rate. In this thesis, we describe our anomaly-based IDS designed for detecting intrusions in cryptographic and application-level protocols. Our system has several unique characteristics, such as the ability to monitor cryptographic protocols and application-level protocols embedded in encrypted sessions, a very lightweight monitoring process, and the ability to react to protocol misuse by modifying protocol response directly.

ACKNOWLEDGEMENTS

This thesis work would not have been possible without the continuous support of my advisor, Dr. Stephen R. Tate. I take this opportunity to thank him for providing me with useful advise throughout my work on this thesis and being always available for discussions. I am grateful to him and also to members of the Computer Privacy and Security (CoPS) lab for discussing the current research topics during weekly research meetings, which helped me in formalizing my thesis reasearch idea. I would also like to thank Dr. Armin R. Mikler for helping me maintain my motivation during the course of this thesis, and for being a very open-hearted, helpful, and enthusiastic person. My thanks also go to Dr. Ram Dantu for being available for discussions regarding my thesis work and advising me on possible future directions. Discussions with him have helped me in getting a different perspective about research.

I also thank Prasanna Iyengar, member of the Network Research Laboratory (NRL), for discussing some of the technical issues with me. My thanks also go to my fiance, Sangeeta, who showed tremendous amount of patience and support during the past one year. I sincerely thank her for being there whenever I needed her. Last but not the least, I am grateful to my parents for supporting me in whatever I wished to do and for being a source of inspiration.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Cryptographic protocols are communication protocols that rely upon cryptography to provide security services across distributed systems. Applications are increasingly relying on encryption services provided by cryptographic protocols to ensure confidentiality and authentication during secure transactions over the network. However, the security provided by these encryption services might be undermined if the underlying security protocol[1] has design or implementation flaws. In fact, research has revealed numerous weaknesses in security protocols [19], [20], [21], [22], [23], [25] with results ranging from the misuse of encryption [30] to compromising the private encryption key [36]. Much research in this area has focused on applying formal methods for analysis and verification of security protocols, and although most of this research was successful in detecting flaws in and improving the protocols, it remains a fact that complete security of cryptographic protocols is still a work in progress. Given the imperfect nature of security protocol design, it can be concluded that in order to be able to improve the confidence in security protocols, detecting intrusions in those protocols is important.

Intrusion detection is a well studied field, and two main detection approaches have been in use for several years: signature-based and anomaly-based. The signature-based approach is effective for detecting known attacks, since it matches the monitored data with a database of known attack signatures to detect suspicious activities. However, due to its reliance on the availability

[1]The terms cryptographic protocol and security protocol are used interchangeably throughout this thesis.

of attack signatures, it is ineffective against previously unknown attacks and modified versions of known attacks. In addition, available but imprecise signatures may cause increased false positives. Anomaly-based detection tries to ameliorate these problems by focusing on modeling normal system behavior in order to be able to detect behavioral deviations, rather than explicitly matching attack signatures. A normal system behavior profile is created by observing data over a period of time, and then intrusions are detected as deviations in the monitored behavior, enabling anomaly-based detection systems to detect novel attacks. Although this is a distinct advantage over signature-based systems, anomaly-based systems have demonstrated difficulty in selecting system features in order to characterize the normal behavior in such a way that any subtle deviation caused by malicious activities is not missed, and at the same time expected deviations do not generate alerts. Further, imprecise normal behavior profiles may increase false alerts, limiting the applications of anomaly-based systems in practice. However, recent research has introduced a new approach, specification-based detection, which has been applied to address the problem of high false positives. The specification-based intrusion detection approach uses manually developed specifications to characterize the legitimate system behavior, rather than relying on machine-learning techniques to learn the normal behavior by observing data, thus eliminating false positives caused by legitimate but previously unseen behavior. The advantages of specification-based and anomaly-based approaches were combined by Sekar et al. [34] in their specification-based anomaly detection system, which greatly simplifies feature selection while being able to detect novel attacks.

Given the success of current intrusion detection technology, applying it for detecting intrusions in cryptographic protocols seems to be an attractive choice. However, it must be noted that most

network intrusion detection systems inspect network packet fields in order to match them with attack signatures or to generate a high-level model of interactions between communicating principals to detect suspicious activity. These activities become infeasible at the network level when protocol sessions are encrypted at the application level, which suggests that application level techniques must be employed in order to detect intrusions in cryptographic protocols and application-level protocols embedded in encrypted sessions. Indeed, recently, Yasinsac [28] demonstrated that dynamic analysis of security protocols, rather than a static analysis as in the case of formal methods, enables detection of certain class of attacks on cryptographic protocols. Yasinsac's technique is based on protocol-oriented state-based attack detection, which reconstructs protocol sessions in terms of state models and matches these with previously generated attack state models to detect attacks. However, the attack behavior is modeled as state-machine representations of execution traces of known protocol attacks, which is essentially a signature-based technique since it needs explicit knowledge of attack sessions, and hence has a drawback of not being able to detect novel attacks.

1.1 ProtoMon

Our technique derives inspiration from the specification-based anomaly detection system of Sekar et al. [34] and inherits its benefits, such as reduced false alarm rate, simplified feature selection and unsupervised learning. In this thesis, we propose ProtoMon, a novel approach of instrumenting shared libraries for cryptographic and application-level protocols to be able to detect and prevent intrusions in those protocols. ProtoMon detects attacks on protocols embedded in encrypted sessions since we integrate the monitoring into processes taking part in the protocols. Monitoring at a gateway or even another process on the same machine will not be able to detect these attacks. The

framework that we propose has an ability to move data collection and analysis off the host to make the performance impact minimal. Also, moving the analysis from the hosts to a central protocol monitor process makes it possible to correlate alerts in order to further reduce the false alarm rate and to detect network-wide attack patterns. We present experimental results to demonstrate the effectiveness of our approach in detecting some of the recent attacks on OpenSSL, such as timing attack and password interception [36], [37].

To start with, we will give definitions of some of the concepts used in this thesis.

1.2 Definitions

Intrusion. An intrusion can be defined as *"any set of actions that attempt to compromise the integrity, confidentiality, or availability of a resource"* [1]. These actions involve unauthorized access or manipulation of the resource either by an outsider who does not have any privileges on the resource or by an insider who exceeds the assigned privileges. Recent studies show phenomenal increase in the number of intrusions as a result of increase in the networked machines: from merely 6 incidences reported in 1988 the number has grown to 82094 in 2002 [2].

Intrusion Detection. Intrusion detection is defined as *"the problem of identifying individuals who are using a computer system without authorization (i.e., crackers) and those who have legitimate access to the system but are abusing their privileges (i.e., the insider threat)"* [3]. As described in section 1, intrusion detection approaches are categorized in two main types: signature-based and anomaly-based. Signature-based detection is effective against known attacks, whereas anomaly-based detection is effective against known and unknown attacks.

Intrusion Detection System. An Intrusion Detection System (IDS) is an application that is responsible for monitoring and analyzing host or network activity to detect intrusions in order to protect information from unauthorized access or manipulation. IDSs are classified in two main types: network-based and host-based. Network-based IDSs use sensors to monitor the traffic in all network segments and collect and analyze the data to detect intrusions, whereas host-based IDSs analyze the traffic destined for or processes running on a particular host. Crosbie and Spafford give a list of properties that are desirable for an IDS [4]:

- It must run continually without human supervision.
- It must be fault tolerant by being able to recover from system crashes, either accidental or caused by malicious activity. Upon startup the intrusion detection system must be able to recover its previous state and resume its previous operation unaffected.
- It must resist subversion. The system must be able to monitor itself to ensure that it has not been modified by an attacker.
- It must impose minimal overhead on the system where it is running to avoid interfering with the system's normal operation.
- It must be configurable according to the security policies of the system that is being monitored.
- It must be adaptable to changes in system and user behavior over time as new applications and resources are added and as users change their activities.

1.3 Secure Sockets Layer (SSL) Protocol

We use the SSL protocol to demonstrate the ability of our proposed system to detect attacks. This warrants some discussion of the workings of SSL. SSL is a widely used protocol which combines the strengths of symmetric and asymmetric key cryptography and aims at providing secure key distribution, authentication, and data transfer services. SSL, which was originally proposed by Netscape®[2], has been expanded and standardized in the latest IETF standard TLS (Transport Layer Security) protocol [5]. SSL protocol, as shown in Figure 1.2, is composed of two layers: the record protocol and the handshake protocol, each one used during different stages of the protocol run. According to the specification, *"The primary purpose of SSL is to provide data integrity and privacy between two communicating applications."* The handshake protocol is used to negotiate a unique symmetric key per session between the two communicating applications, which is then used by the record protocol to provide data encryption during data transfer.

1.3.1 SSL/TLS Handshake Protocol

SSL provides a handshake sub-protocol that uses public-key encryption techniques to enable the communicating applications to authenticate themselves and negotiate security parameters before they can start a secure data transfer at the record layer. Figure 1.1 shows various messages exchanged between a server and a client during the handshake. At the end of the handshake, the two peers generate the same master secret and verify that they have agreed upon the same security parameters. The handshake also allows resumption of a session that has been legally terminated

[2]Netscape Communications Corporation

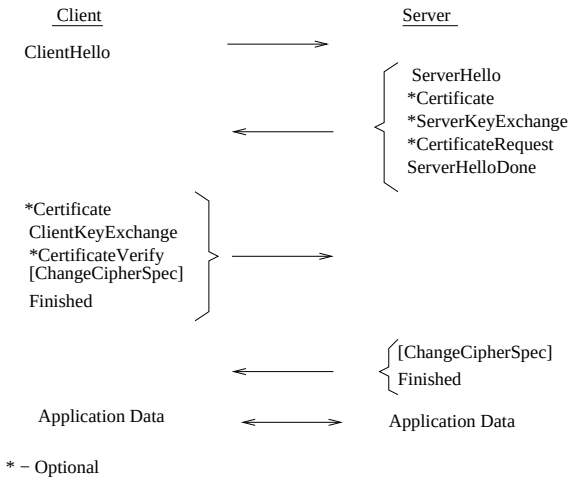

Figure 1.1: SSL Handshake

and whose session id is still in the server cache. The *change cipher spec* message is not a part of the handshake protocol and is specified as a separate sub-protocol which allows the peers to signal change in ciphering strategies. Another sub-protocol, the *alert* protocol, is of a particular importance in the context of one of the attacks on OpenSSL described later in this thesis. The *alert* protocol is used to communicate errors and fatal alerts that result in an immediate termination of the connection, making that session non-resumable. Alerts could be generated because of failures at different stages in the computation of secrets, authentication of messages, and decryption of the data. One of the attacks detected by our system takes advantage of the difference in the time it takes to generate two different alerts: *bad_record_mac* and *decryption_failed*.

1.3.2 SSL Record Protocol

The SSL record protocol provides data fragmentation, compression, encryption, and transmission services on the transmitting side, and data reassembly, decompression, decryption and delivery

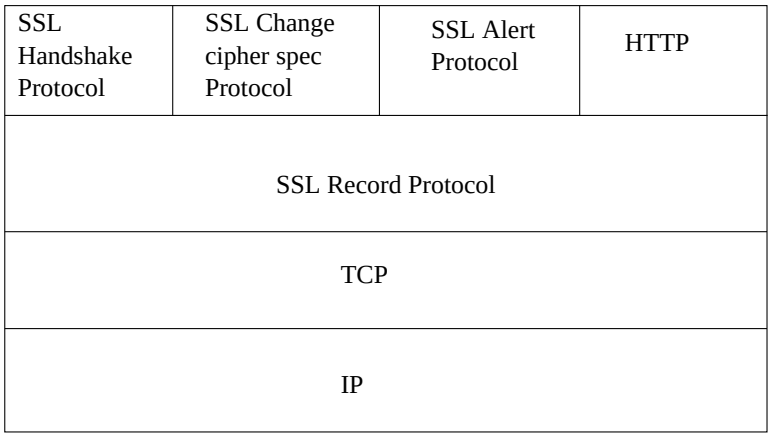

Figure 1.2: SSL Protocol Stack

services on the receiving side. It uses session keys negotiated during the handshake to encrypt and decrypt the data. The record protocol defines a content type, which can be handshake protocol, alert protocol, change cipher spec protocol, or application data protocol.

1.4 Related Work

Anomaly-based IDSs do not compare well with misuse-based IDSs in the production environment due to their high false positive rate, which is a result of the difficulty in characterizing the normal system behavior. However, the anomaly-based approach remains effective for detecting novel attacks and modifications of known attacks. This technique has been studied with various approaches, with the probability-based approach being the most common. Probability-based anomaly detection builds a database of reference network packets with various packet parameters and probabilities of occurrences of those packets. If any of the packets observed in the monitored traffic has the probability below the threshold it is most likely a malicious packet and is flagged as anomalous.

For example, SPADE [6] has a model with four different probabilities:

- P(destination-address, destination-port)
- P(source-address, destination-address, destination-port)
- P(source-address, source-port, destination-address, destination-port)
- Bayes network approximation of the above

ADAM [7] and NIDES [35] are also probabilistic anomaly detection systems which use frequency-based statistical models of normal network traffic, in which probability of an event depends on frequency of that event during the training period. Time-based probabilistic approaches are implemented in PHAD [8], ALAD [9], and LERAD [10], in which probability of an event depends on the time of its last occurrence. Although the probabilistic approach is the most well studied, recent research suggests a focus on more advanced methods such as clustering approaches and state-based detection.

Cluster analysis is a multivariate technique and has been shown to be useful for anomaly detection. Taylor and Alves-Foss proposed NATE (Network Analysis of Anomalous Traffic Events), which uses cluster analysis to form normal groups of TCP/IP sessions based on Euclidean distances between TCP flag and byte counts [11]. Attack sessions are then detected as data points that do not fall in any of the normal groups. More recently, Arshad and Chan proposed the CLAD (CLustering for Anomaly Detection) algorithm that finds suspicious clusters as outliers, which are clusters far away from and of size unusual to the normal clusters [12]. The cluster size is defined in terms of cluster width, which specifies a local neighborhood of clusters that are considered close. However,

these clustering techniques use data sampling to form normal clusters by selecting network packet fields as data points, which is impossible when the sessions are encrypted.

Machine-learning is another approach to anomaly detection that learns normal behavior of a subject[3] in terms of sequences of actions performed or commands executed by that subject. Lane and Brodley proposed a machine-learning approach that is based on a hypothesis that a user responds in a similar manner to similar situations, leading to repeated sequences of actions [13]. These characteristic sequences of commands generated by the user are then used to form a user profile and serve as a fundamental unit of comparison in their anomaly detector. Forrest et al. have developed an anomaly detection system based on human immunology that monitors sequences of system calls generated by privileged processes (daemons) to compare them against profiles created during learning phase [14]. Our approach is similar in that we also use machine-learning technique to characterize normal usage of cryptographic and application protocols.

Lee and Stolfo proposed an application of data mining to intrusion detection, in which normal usage patterns are formed by learning (mining) large amounts of audit data [15]. As noted by them, computing the rule sets in order to learn the normal behavior with the help of data mining is certainly not a lightweight process, making it unsuitable for real-time intrusion detection unless a distributed detection framework is designed. By contrast, our approach is inherently lightweight, enabling it to detect attacks on cryptographic protocols without excessive computing power. In addition, it has the ability to move the data collection and analysis off the host to make performance impact minimal and to enable centralized alert analysis and correlation.

[3]A subject can be a user, a program, or a protocol

Statistical-based anomaly detection relies on collecting normal usage statistics of an object during the training phase to create statistical models. It is important that the training data be free of any attacks so that statistics generated by attacks are not included in the long term normal usage profile. Anomalous activity is detected by quantifying observed statistics over a number of variables to create a short term profile and comparing it to the long term profile. If the difference in these two profiles is greater than a predetermined threshold then that short term activity is flagged anomalous, otherwise it is included in the long term profile to adapt to the changing system or user behavior. SRI's NIDES/STAT [35] algorithm uses similar approach and uses "chi^2" [16] like tests to determine the similarity between short-term and long-term profiles.

All of the approaches for anomaly-detection discussed above have one common problem: choosing the correct features that need to be learned in order to create an accurate normal behavior profile of the system. Consequently, attacks that manifest themselves as anomalies in the features not included in the normal behavior profile are likely to be missed. This problem is ameliorated by a new variant of anomaly detection, specification-based anomaly detection, which uses specifications of protocols given in standard documentation like RFCs to be able to select appropriate features. Use of protocol specifications enables manual characterization of a legitimate behavior rather than learning the normal behavior by observing data over a period of time. As another advantage, this approach alleviates the problem of high false positives evidenced in the anomaly detection, by eliminating the false positives caused by behaviors that are legitimate but absent in the training data.

Sekar et al. proposed a method, specification-based anomaly detection, that uses state-machine specifications of network protocols and statistical learning to map packet sequence properties to properties of state-machine transitions to detect network intrusions [34]. They model the network protocols using extended finite state automata (EFSA) to be able to divide up packet sequences into traces, where each trace corresponds to a path in the state machine. The specification language proposed by them enables capturing all the relevant information about the state-machine and the statistics that need to be maintained to detect the attacks. The effectiveness of their approach is evidenced at a network gateway by successful detection of most of the attacks by simply monitoring the distribution of frequencies with which each state transition is taken. However, as noted by Sekar et al., current intrusion detection techniques rely primarily on inspecting network packet fields to gather information about the system behavior, which becomes infeasible when sessions are encrypted making it difficult to detect intrusions. With the advent of IPv6 and wider use of IPSec this problem will get worse, as there will be more end-to-end encrypted connections making network-level detection techniques ineffective for detecting application-level attacks. Our approach addresses this problem by embedding the monitoring into the protocol process, thus eliminating the need for inspection at the network level.

Encryption is used by underlying cryptographic protocols to distribute keys and authenticate principals and data over the network. Unfortunately, more than two decades of research has demonstrated that designing secure protocol is an extremely difficult problem and as a result many security protocols have serious design and implementation flaws. Extensive work has been done to formally verify and test the security protocols to prove that they are secure. Several works have

been published on the topic after Needham and Schroeder [18] first suggested the possibility of applying formal methods for cryptographic protocol analysis, and Dolev and Yao [19] developed a polynomial-time algorithm for deciding the security of restricted class of protocols. Based on these works, several tools for formally analyzing security protocols were developed including special-purpose model-checkers such as the Interrogator [20], and the NRL Protocol Analyzer [21], and general-purpose model-checkers such as FDR [22] and Mur [23]. Most of the tools were successful in detecting flaws that had been previously undetected, as in the case of the NRL protocol analyzer, which uses inductive theorem proving techniques and was able to find a flaw in the Simmons Selective Broadcast Protocol [21]. However, they exhaustively search through all possible sequences of actions to check if the attack could be possible, and hence suffer from state space explosion. Song proposed Athena [25], an extension of the strand space model [24], employing state space reduction techniques to be able to verify security properties of a protocol for an unbounded number of concurrent runs. One point that must be noted about all of the formal verification techniques discussed so far, is that they use mathematical models of protocols in a laboratory environment to perform symbolic protocol runs in order to prove the security properties. However, given that the protocol security problem is undecidable [26], [27], theoretical techniques have limited application to the problem, meaning that the analysis tools will not be successful all the time. Thus, supplemental to the formal methods, a mechanism for detecting misuse of cryptographic protocols is highly desirable in the production environment. This argument is further strengthened by recent research results that continue to uncover serious flaws in cryptographic protocols [36], [37]. We address this problem by proposing a way to monitor the behavior of cryptographic and application-level

protocols during their execution, unlike the off-line formal analysis approach, to detect intrusions.

More recently, this issue was addressed by Yasinsac who applied classic knowledge-based and behavior-based intrusion detection techniques for detecting intrusions in security protocols [28], [29]. This technique develops signatures of attacks on protocols by gathering information from three sources:

- Known attacks identified in the literature.
- Attack taxonomies identified in the literature.
- Flaws and suspicious activity gathered during execution.

The attack signatures are developed from protocol execution traces modeled as state machines, which are then used by the detection system for matching with the state machines that are dynamically built during actual protocol execution. If the observed state transition sequence is similar to one of the signature state transition sequences then it signals the probability of an ongoing attack. Unlike our approach, this approach requires explicit knowledge of attack signatures in terms of protocol execution traces making it a signature-based system. Our approach needs no prior knowledge of attack signatures, as it detects attacks as behavioral anomalies, thus enabling it to detect novel attacks as well as modifications of known attacks.

CHAPTER 2

OVERVIEW OF PROTOMON

Identifying protocol implementation libraries that participate in the protocol state machine is our first step for instrumenting embedded monitors in the protocol process. We implement the embedded monitors as shared libraries that are integrated with other cryptographic libraries in an implementation, and that act as sensors within our monitoring framework. However, it should be noted that unlike traditional IDS sensors which extract data from passive sources such as audit logs, our sensors are directly integrated inside the process and generate signals as the process executes.

2.1 Generic Protocol Monitoring Framework

Figure 3 shows the basic architecture of our system, which we describe more fully in the remainder of this section. We propose a generic protocol monitoring framework that can be configured as a host-based intrusion detection system or as a central protocol monitor for all hosts on a network, moving data collection and analysis off the hosts to minimize the performance impact, as illustrated in Figure 2.1. Our framework consists of three main components: the protocol monitor, pluggable protocol behavior profiles, and monitor stubs. We now briefly describe each of these components.

2.1.1 The Protocol Monitor

The protocol monitor is primarily responsible for collecting and analyzing the protocol state transition notifications sent by the stubs and detecting anomalies in the protocol behavior. The anomalies

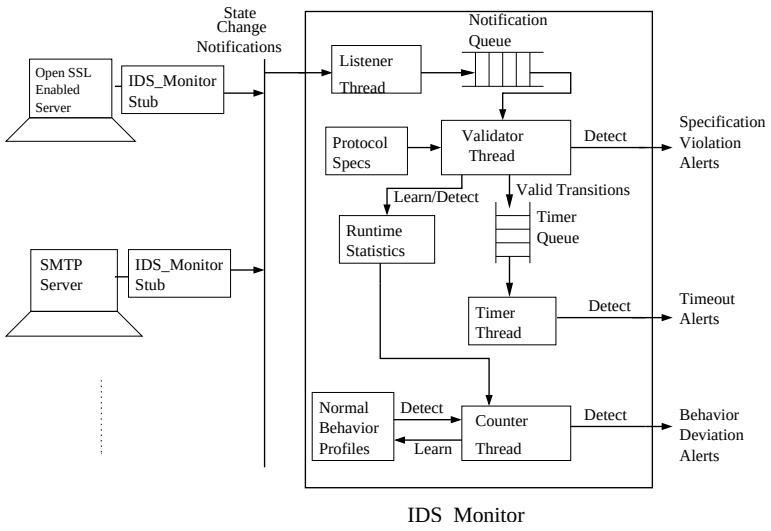

Figure 2.1: Generic Protocol Monitoring Framework

are detected by building a short-term profile of the protocol usage and comparing it with the normal behavior profile built during the learning phase. We show in section 4 that most of the attacks that are within the scope of our system are detected by monitoring the state change notifications. The protocol monitor can be started to operate in three modes: learn, detect, and prevent, which are described next.

Learn mode. In the learn mode, the protocol monitor creates normal behavior profiles for target protocols by taking manually developed protocol state-machine specifications and then observing training data for a period of time to be able to perform statistical analysis of state transitions reported by monitor stubs. It is important that the training data reflects the expected usage pattern of the protocol so that the resultant normal behavior profile captures the correct expected behavior, thus reducing the false positives. However, it should be noted that since we use the specification-based approach, the scope of the learning phase is limited to statistical analysis of

legitimate protocol use, rather than learning the legitimate use itself which can cause false alarms generated by legitimate behavior that is unseen in the training data. Further, generally the use of the network services, and hence the use of the protocols, is much more during business hours compared to the use during non-business hours. In order to reflect this variance in the protocol usage pattern during different times of the day, we divide each 24 hour period into six different time slots: midnight 00:00:00 to 03:59:59, early morning 04:00:00 to 06:59:59, morning 07:00:00 to 11:59:59, afternoon 12:00:00 to 16:59:59, evening 17:00:00 to 19:59:59, and late evening 20:00:00 to 23:59:59. The statistics collected by the monitor in the learn mode are:

- Maximum number of protocol sessions for each time slot.
- Maximum number of broken protocol sessions for each time slot.
- State transition statistics. These include the average of the number of times each state transition is taken. The averages are calculated per second, per minute, and per hour to be able to detect attacks that create sudden and short-term anomalies as well as attacks that cause slow and relatively long-term anomalies.

At the end of the learning phase the protocol monitor creates a normal behavior profile for each monitored protocol by recording and averaging the statistics mentioned above.

Detect mode. The normal protocol behavior profiles created in the training phase are subsequently used by the protocol monitor in the detect mode as a protocol usage baseline, both is terms of legitimate protocol use and expected protocol usage statistics. We define a tolerance limit for the deviation of protocol behavior as the maximum deviation from the normal behavior that

is acceptable and is not flagged as anomalous. The protocol monitor creates a short-term protocol usage profile at regular intervals by observing the data and comparing it with tolerance limits of the baseline provided by the normal behavior profile to be able to detect specification violations and statistical anomalies. To be able to successfully execute an undetected attack, the attack must not create any protocol behavior anomalies and must strictly follow the protocol state transition specifications. Any attempts that cause specification violations or behavior anomalies are immediately detectable.

Prevent mode. Intrusion response is a well-studied field and various approaches are currently in use, with the most common using techniques such as blocking traffic from offending IP addresses and forcefully resetting connections [33]. However, these techniques could be used by an adversary to make an IDS block the traffic from non-offending IP addresses causing a denial of service. We address this issue, as it pertains to responding to protocol misuse, by using the prevent mode of operation. In the prevent mode, upon detecting a protocol behavior that falls above the upper tolerance limit, the protocol monitor coordinates with the monitor stub and slows down the protocol response. This is possible because of our positioning of the monitor stubs inside the protocol process, which allows the monitor stub to introduce a delay in each protocol state transition as long as the protocol monitor signals anomalous behavior. This is similar to some of the other approaches, such as introducing system-call delays in processes that show abnormal behavior [31] and deliberately slowing the connection from suspected scanner machines [32]. Our embedded stubs force the protocol behavior to remain within the upper tolerance limit causing the attacks that need several hundreds of thousands of sessions to be slowed down to the point where they are no longer effective.

Further, elongating the time needed for the attack to be successful allows for human intervention. Also, as a supplemental response to the detected attack, alerts generated by the protocol monitor could be used to invoke other response mechanisms, such as at the router, to stop an attack. Since the stubs remove the delays once the behavior returns back in the tolerance limit, the protocol is not completely halted and our mechanism cannot be used by an attacker to cause an indefinite denial of service. However, it should be noted that the prevent mode introduces increased network traffic overhead due to the increased communication between the protocol monitor and the monitor stub, as illustrated in Chapter 3.

Architecture of the Protocol Monitor. The protocol monitor process consists of four threads: listener, validator, timer, and counter. Figure 2.1 shows the interaction between these threads and the alerts that they generate. The listener collects the state change notifications generated by the monitor stubs and inserts them into a notification queue. The validator thread picks up the notifications from the notification queue and validates them against the protocol specification to detect any specification violations. It also generates statistics of protocol usage which are later used by the counter thread to create a behavior profile. Valid state change notifications are inserted in the timer queue for timing each state. Timing each state allows the protocol monitor to generate timeout alerts if the protocol state machine is aborted before the final state is reached. Observing the number and frequency of timeout alerts detects some of the side channel attacks described in Chapter 3. The counter thread constantly compares the normal behavior profile with the short term profile, referred to as "*Runtime statistics*", that it generates using the state transition statistics, collected by the validator thread. Significant difference between the state transition statistics in

the short term profile and the ones in the normal behavior profile are detected as anomalies by the counter thread.

2.1.2 Pluggable Protocol Behavior Profiles

At the end of the learning phase, protocol behavior profiles are generated for all target protocols. A behavior profile has a specification component, which is manually developed in terms of protocol state-machine by studying implementations of the protocol and standard documents like RFCs. The specification component is complimented by a statistical component, which is built during the training phase, in which training data is used to learn normal protocol usage statistics for six different time periods in a day, with one each for weekdays and weekends. The specification component characterizes legitimate protocol usage in terms of valid states, start states, final states, and all valid state transitions. The rationale behind incorporating these two components is that, some of the attacks cause the protocol to directly violate the protocol specification and are thus detected immediately by validations performed against the specification component, whereas attacks that may not generate specification violations but merely are manifested as traffic or usage anomalies are detected due to the usage baseline provided by the statistical component. It is important that the training data resembles the use of the protocol in the target environment to ensure increased precision of behavior profiles. Profiles generated in this way are then loaded in to the protocol monitor and are used to be able to enforce legitimate and expected protocol usage and to detect attacks as anomalies.

The following is a sample entry in the profile of OpenSSL for a weekday afternoon:

```
Statistics weekday afternoon{
  8464 {8496 1 64 3779}
  8496 {8512 1 64 3779} {8656 0 0 0}
  8512 {8528 1 64 3779}
  8528 {8544 1 64 3779}
  8544 {8560 1 64 3779}
  8560 {8448 1 64 3779}
  8448 {8576 1 64 3779} {8640 0 0 0} {3 1 64 3779}
  8576 {8592 1 64 3779}
  8592 {8608 1 64 3779}
  8608 {8640 1 64 3779}
  8640 {8656 1 64 3779} {3 0 0 0}
  8656 {8672 1 64 3779}
  8672 {8448 1 64 3779}
  3
  maxSessions=9983
  maxBrokenSessions=57
}
```

Each line begins with a number of a current state followed by all states reachable from that state and the statistics on that particular state transition for three different time intervals: 1 second, 1 minute and 1 hour. This enables detection of attacks that create short-term anomalies which might

not get manifested as an anomaly in relatively long term statistics, as well as attacks that create anomalies in the long-term statistics.

2.1.3 Monitor Stubs

Stubs provide a generic interface between the protocol process and the protocol monitor and are integrated into shared libraries of the protocol implementation. Integrating the stubs with the protocol libraries allows them to capture the protocol activity dynamically, rather than relying on passive methods such as audit logs, and to monitor encrypted protocol sessions. Monitor stubs perform a handshake with the protocol monitor to check network availability and to know the mode in which the protocol monitor is running (learn, detect, or prevent). The stubs send protocol state change notifications to either the local protocol monitor functioning as a host-based monitoring system, or to a central protocol monitor for network-wide analysis and to reduce the performance impact on the host. During the experiments, the process of integrating the stubs with the protocol libraries required a very minimal change in the protocol implementation files since the state transitions were clearly and directly implemented in the original libraries, which we conjecture would be the case with any well-written protocol library.

2.2 Benefits of ProtoMon

- *Detection of attacks on cryptographic protocols and application-level protocols embedded in encrypted sessions.* As a primary contribution of our work, our system is able to detect attacks on the application protocols that use encryption to provide authentication, key distribution,

and other services necessary for secure communication between participating principals. Current intrusion detection systems which detect attacks on the application-level protocols primarily analyze the application-level payload in network packets to match patterns of known attacks. Use of encryption by the application-level protocol makes this analysis infeasible since the payload is encrypted and can only be decrypted at the application-level after applying proper decryption operation. We have addressed this issue by positioning our dynamic analysis mechanism inside the application-level protocol process. Further, application of the specification-based anomaly detection approach enables detection of known and unknown attacks and should keep the false positive rate low.

- *Generic framework.* Although we demonstrate the ability of our system to detect attacks on application-level and cryptographic protocols, the design of our framework is generic, meaning it could be applied to any arbitrary protocol with clearly defined specifications. As noted earlier, this will allow our system to act as a central protocol monitor for the entire network enabling correlation of alerts generated due to behavior deviations in protocols running on different hosts.

- *Response ability.* Intrusion response systems aim at preventing or minimizing the damage caused by detected intrusions. Detection systems that are based on analyzing information from passive sources such as audit logs rely on peripheral response actions such as isolating the target host or changing network firewall rules to block traffic from offending IP addresses. However, the response of an application which is under attack does not change to defend itself. In our approach, due to their positioning inside the protocol process, the monitor stubs

enable changing the response of the protocol that is being attacked.

- *Lightweight.* Our system is lightweight meaning it adds minimal overhead to the host that is running the monitored protocol. The prevent mode introduces increased overhead, as compared to the detect mode, due to the increased communication between the monitor and the stub.

CHAPTER 3

EXPERIMENTAL RESULTS

For evaluation purposes we used attacks against OpenSSL.

3.1 Experimental Setup

We carried out the experiments on a private LAN with no considerable background traffic and with the configuration shown in Figure 3.1. During the learning phase, normal OpenSSL protocol usage was simulated by using a four week access log of a real web site. Any instances of attacks in the access log were removed in order to exclude the behavior of the protocol during the attack. Statistics collected during the learning phase were then used in the detect phase to demonstrate the ability of the system to detect attacks not observed during the learning phase.

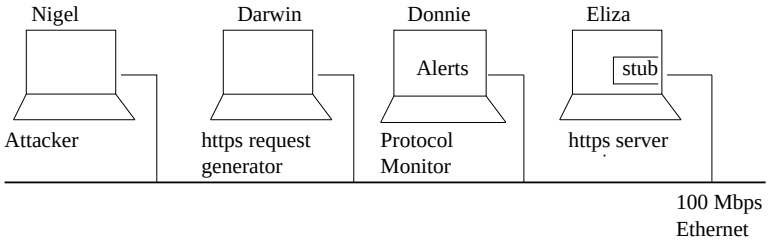

Figure 3.1: Experimental setup

3.2 Attack Prevention

We categorize attacks that are within the scope of our system into two general classes: *atomic*, and *non-atomic*. We denote attacks which need to use the protocol once or for very few times to be

successful as atomic. On the other hand, attacks that are based on using the protocol for a large number of times are referred to as non-atomic. Non-atomic attacks are preventable in the prevent mode of the protocol monitor, whereas atomic attacks are only detectable. This follows from the fact that we use an approach of slowing down the protocol response if the protocol monitor detects the protocol behavior going beyond the upper tolerance limit. Atomic attacks do not generate this type of anomaly and hence cannot be prevented by the slow-down mechanism. However, our results show that this mechanism is effective in considerably elongating the time taken by an ongoing non-atomic attack to complete. The *slow-down-factor* can be tuned to a value that practically prevents the attack from becoming successful. In the following section, we show the results of performing three specific attacks on OpenSSL.

3.3 Attack Detection

1. *Side-channel attacks.* Side-channel attacks have been shown to be possible and practical on cryptographic protocols based on RSA encryption routines [36], [37], [38], [39]. These attacks deduce the private key, or invert the encryption with the help of information that is unintentionally leaked by the protocol. In the case of timing attack against OpenSSL implementations, it takes around 350,000 failed session negotiation attempts to break a 1024-bit RSA private key [36]. As a general characteristic of these attacks, each failed attempt to negotiate a session results in aborting the protocol state-machine before the final handshake state is reached. Our protocol monitor times each state and observes the increased number of timeouts to be able to detect these attacks. We simulated the timing attack by tweaking the

OpenSSL client library and using a dummy SSL client application to repeatedly abort the handshake at a particular state. Figure 3.2 demonstrates the increased number of timeouts during the time when the attack is in progress. The attack simulation took approximately 6 hours to complete in the detect mode, whereas in the prevent mode, with 1 second delay introduced in the state transitions after the threshold number of sessions were aborted, the stub slowed down the protocol and it took an average of 7 seconds per session. As seen, this considerably elongates the time taken for the attack to be successful. While the original attack simulation completed all 350,000 probes in approximately 6 hours, the prevent mode protocol monitor allowed only about 15,420 probes in the first 30 hours.

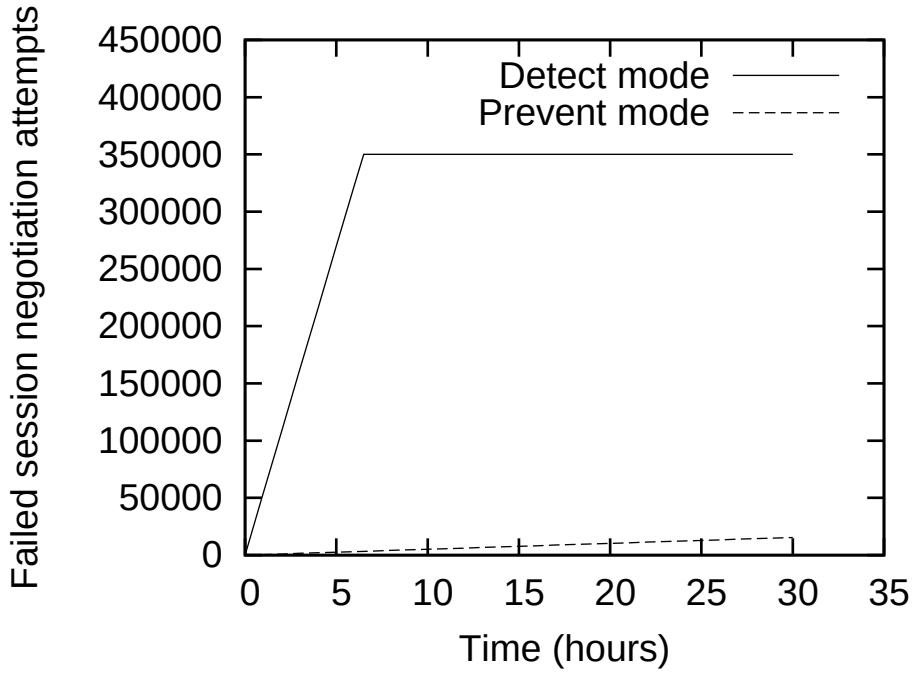

Figure 3.2: Detection of Timing Attack on OpenSSL

2. *Rollback Attacks.* Wagner and Schneier pointed out possible vulnerabilities in the specification of SSL version 3 [40], and we specifically demonstrate detection of version rollback and cipher-suite rollback attacks. SSLv3 is susceptible to a version rollback attack in which the *client hello* message sent by the client during session resumption can be intercepted by an intruder to change the version number to 2, thus forcing the server to downgrade the version used in the session with that client. If the server finds the session id supplied in the client hello message in its cache, it assumes resumption of a previous session, but with a lower SSL version and directly proceeds to the *finished* message. Further, unlike SSLv3, the *finished* message in SSLv2 does not include the version number making this attack undetectable. The downgrading of the SSL version by the server exposes the server to several vulnerabilities in SSLv2. To be able to detect this attack, the protocol monitor maintains separate session caches for SSLv2 and SSLv3 as suggested by Wagner and Schneier. This attack is detected by simply observing a state transition from the *SSL2_ST_SEND_SERVER_HELLO_A* state to the *SSL2_ST_SERVER_START_ENCRYPTION* state reported by a monitor stub embedded in SSLv2 for a session id that was previously registered with the protocol monitor by a stub in SSLv3.

   The cipher-suite rollback attack exploits the fact that the *change cipher spec* message is excluded from the calculation of a symmetric MAC on the previous handshake messages, which is used as an authentication code for the *finished* message. The attacker can drop the *change cipher spec* message causing the server and the client to never change the pending cipher-suite to the current cipher-suite, potentially disabling encryption and message authentication at

the record layer. Although the SSLv3 protocol specification documents the *change cipher spec* message as an optional message, it is most likely a cipher-suite rollback attack if this message is excluded from the initial handshake since the initial configuration provides no encryption or authentication, with the exception of the session resumption scenario. The protocol monitor requires the *change cipher spec* message during the initial handshake and dropping this message by the attacker results in a specification violation that is immediately detected. Also, note that, unlike signature-based systems, the monitor has no prior knowledge of the attack.

3. *Buffer overflow and Denial of service attacks.* Buffer overflow vulnerabilities that exist in the implementations of SSL protocol routines [42], [41] could be exploited to execute arbitrary code on the server. For example, an Apache/mod_ssl worm exploited a buffer overflow during the SSLv2 handshake process. Although the attacker can gain full control of the server by executing this arbitrary code, it should be noted that the attacks that fall in this category most likely result in aborting the state-model of the protocol at an arbitrary state. The protocol monitor detects this behavior by using the timeout mechanism described earlier. The timeout alert could then be correlated with the anomaly in protocol statistics reported to the protocol monitor to detect the denial of service that may be caused by the executed code.

3.4 Behavior Comparison

Figure 3.3 demonstrates the behavior of OpenSSL during the two operating modes: *detect* and *prevent*.

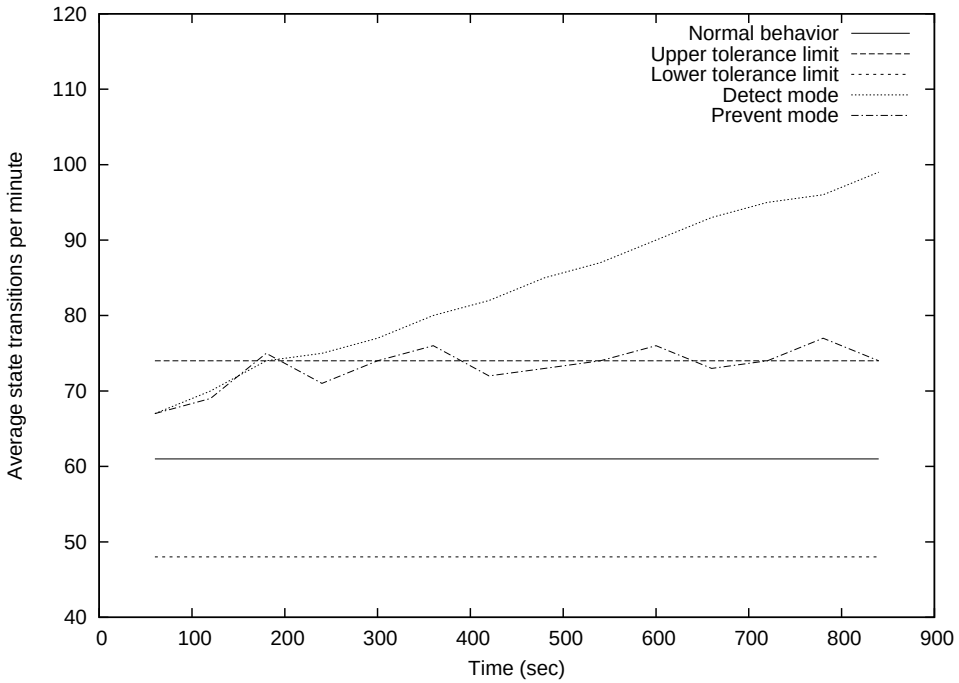

Figure 3.3: Protocol behavior comparison

As seen, alerts are generated in both modes when the protocol behavior deviates from the normal behavior beyond the tolerance limit. However, in the prevent mode, the protocol monitor signals the monitor stub to insert delays in the protocol state transitions upon detection of significant deviation, effectively slowing down the protocol and forcing the protocol behavior to be within the upper tolerance limit of the normal behavior. If the protocol behavior deviates towards the lower tolerance limit, alerts are generated in both the modes.

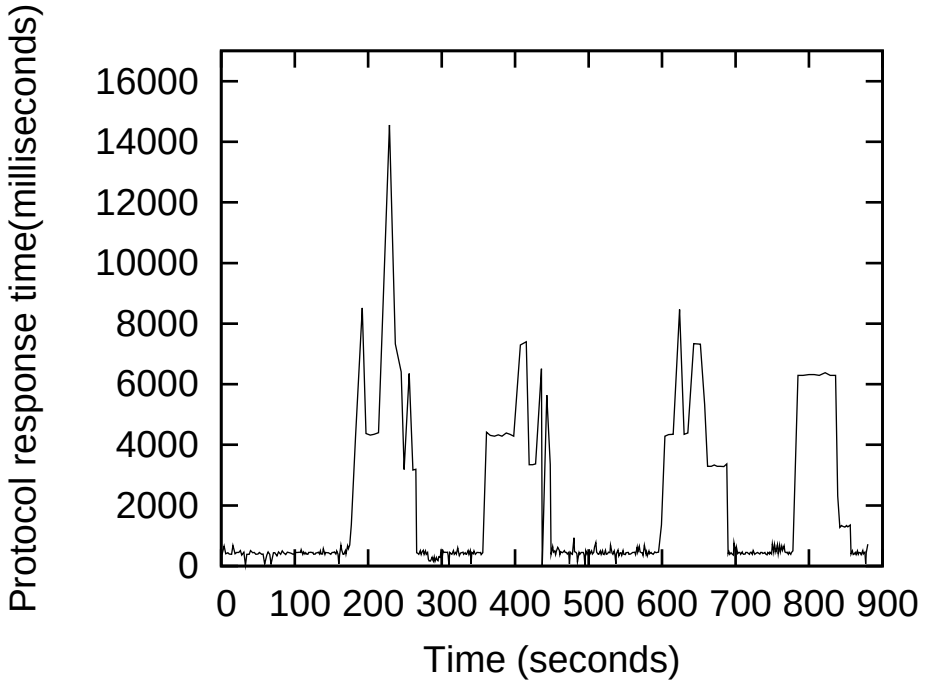

Figure 3.4: Protocol response time variation

Figure 3.4 illustrates the variation in the protocol response time due to the delays introduced by the stub after the protocol monitor signals that the deviation of the protocol behavior is beyond the upper tolerance limit. The behavior is sampled every second, every minute and every hour. Figure 3.3 shows the behavior sampled every minute with the behavior crossing the upper tolerance limit at the beginning of the fourth minute. As a result the protocol response time plotted in Figure 3.4 shows the corresponding increase for the period of approximately 90 seconds.

3.5 Performance Overheads

Given that cryptographic computations based on encryption standards such as RSA are complex and resource intensive, it is desirable that the overhead added by any mechanism that aims at

detecting intrusions in protocols based on these encryption standards is to the minimum. This section shows overheads in the performance of the OpenSSL protocol implementation caused by our detection and prevention mechanism, and measured with our experimental setup of Figure 3.1.

3.5.1 Network Bandwidth

	Detect mode	Prevent mode
Network overhead per SSL session	840 bytes (14.7%)	1162 bytes (20.4%)

Table 3.1: SSL sessions vs. network traffic overhead

The monitor stub sends state transition notifications to the protocol monitor over the network when they are running on different hosts, resulting in an overhead in the network traffic. Table 3.1 shows the average network traffic overhead over 1000 SSL sessions with average of 2465 bytes of SSL protocol related data and average of 3243 bytes of application data, all measurements taken on the wire. For each state transition reported to the protocol monitor by the stub, the actual UDP payload is only 12 bytes (3.9% overhead), with the rest of the overhead being the Ethernet, IP, and UDP header information. It is possible to collect the state transitions and pack them in a single notification in order to reduce this network overhead; however, we have not experimented with this option. It is also possible to send the state change notifications on a separate network interface for heavily used servers to reduce overhead on the production network.

3.5.2 Protocol Response Time

	Response time per request(miliSec)	Percentage slow-down-factor
OpenSSL	385.66	-
Learn mode	404.81	4.9%
Detect mode	406.02	5.2%
Prevent mode	2080.60	439.4%

Table 3.2: Response time comparison

As described in Section 3.1, we used the normal behavior profile of OpenSSL to detect anomalies in the detect and prevent modes of operation. Table 3.2 shows average response time of the OpenSSL enabled web server per request after anomalies have been detected. Prevent mode slows down the protocol response by the factor of approximately 4.4 for the purpose of this experiment. This *slow-down-factor* can be tuned to a desired value by adjusting the delays introduced by the monitor stubs. Consequently, the time taken by any attack that spans across multiple protocol sessions can be elongated when the protocol monitor is running in the prevent mode.

3.5.3 Processor

	Processor usage time per session (milliSec)	Percentage overhead
Standard OpenSSL	22.2	-
Learn mode	25.5	14.8%
Detect mode	25.7	15.7%
Prevent mode	26.3	18.4%

Table 3.3: Processor usage time comparison

When the protocol monitor is used as a host-based detection mechanism it uses the processor time on the server. Our experiment involving measurements of processor overhead were carried out by using the protocol monitor as a host-based detection system. We used a host with a Pentium4 2GHz processor to run the protocol and the protocol monitor. Table 3.3 gives a percentage increase in the processor usage by the protocol process when the protocol monitor is used in learn, detect, and prevent mode. This overhead can be minimized by moving the protocol monitor process to a central host to perform a network-wide alert analysis and correlation.

CHAPTER 4

DISCUSSION

Inspite of the substantial amount of research that has been performed on problems in detecting intrusions accurately, intrusion detection is still a work in progress. Consequently, relatively recent paradigms, such as intrusion response, intrusion tolerance, and survivable systems need wider consideration as mechanisms supplemental to passively detecting intrusions. In this chapter we consider open research problems and challenges to both intrusion detection in general, and with respect to the specific approach explored in this thesis.

4.1 Current Intrusion Detection Issues and Challenges

1. *Automatic, accurate, comprehensive and faster intrusion response mechanisms.* The recently observed increase in the speed at which the Internet worms are spreading has brought manual intrusion response mechanisms requiring human intervention to the point that they have started to become less effective for keeping the damage at the minimum and stopping the worms from spreading to other networks or hosts. Although, containing such attacks or worms would need faster and automatic intrusion response schemes, it remains a fact that further research is needed in order to make these response schemes more reliable and accurate to prevent the attacker from using such a mechanism against the network or the host that is being protected.

2. *Feasibility of continued use of current IDS technologies under IPSec/IPv6 environments.* We recommend that the current IDSs should be studied to evaluate the feasibility of their continued use in the IPSec/IPv6 environments. IP level encryption is being considered as a mechanism to prevent attacks that are based on mangling IP packets on the network. It should be noted that given the heterogeneous nature of the attacks, it is unlikely that this mechanism will be able to address all the attacks, suggesting that the IDSs will still be required to detect intrusions. However, current IDS technology is mainly based on analyzing network packets to detect malicious activities. Consequently, such an analysis will become impossible under IPSec/IPv6 environments because of the use of encryption.

3. *Increasing survivability of network applications and services by making them aware of their behavior.* The concept of "Defense in Depth" deals with increasing the reliability and fault tolerance of a security mechanism by creating multiple layers of heterogeneous security mechanisms around a resource that needs to be protected. In the case of a network service, we recommend that the service should be made aware of its normal behavior in the target environment and should be given an ability to act on its own if it detects any abnormal use of the service. We believe that increasing the survivability of the network services or applications in such a manner would help in an effort to ensure an uninterrupted deliverance of services, although with a downgraded performance.

4.2 Future Work on Embedded IDS Monitors

Although our results indicate that our approach of dynamically learning the normal behavior of a cryptographic protocol by embedding monitors inside the executing protocol process detects and prevents attacks on those protocols, there remains a large number of open research issues.

1. It is possible to apply more advanced learning techniques and models such as Markov chains for building comprehensive behavior profiles. Further research is needed in applying such models to learn transition probabilities in a protocol state machine to enhance the precision of the normal behavior profile. Adaptive learning strategies could be employed to be able to continuously modify the normal behavior profile to take into account legitimate and expected changes in the protocol usage patterns.

2. The high false alarm rate experienced by anomaly-based intrusion detection systems is a current research topic. We believe that alerts caused by protocols running on various nodes in the network could be correlated to reduce false alarms generated by a centralized protocol monitoring process. This argument is strengthened by the fact that a successful attack executed by an active attacker often involves compromising various network services on the network to maximize the damage. Consequently, the protocols used by these network services, if monitored at a centralized protocol monitor, should cause alerts that could be correlated for a more comprehensive and correct attack detection.

3. Our approach of responding to the detected anomalies by slowing down the response of an abnormally behaving protocol slows down all clients connecting to that server until the

protocol behavior returns within the tolerance limit of its baseline. We are investigating into a selective response scheme which identifies and slows down the protocol response for only the clients causing the protocol anomalies.

4. As discussed in Chapter 3, our current system is able to prevent non-atomic attacks on the cryptographic protocols. Further research is needed in order to be able to include the atomic attacks in the scope of the response mechanism proposed in this thesis. It could be possible to implement the monitor stub as a wrapper around the protocol process, validating all the state transitions before passing them on to the protocol process. This should allow immediate response to atomic attacks; for example, the cipher suite rollback attack could be prevented by first validating the state transition to detect the dropping of the *change cipher spec* message and then aborting the session to prevent the unprotected session from continuing.

5. During the experimental evaluation of ProtoMon, we tested its ability to detect and prevent attacks by generating attack simulation tools and data specific to our requirements in order to characterize the normal protocol behavior and detect anomalies caused by the attack behavior. Research work is needed in developing a generic and comprehensive test suite for evaluating mechanisms aimed at detecting attacks on protocols embedded in encrypted sessions. Such a test suite would provide a testing baseline making it possible to compare different detection mechanisms on a generic evaluation criteria.

4.3 Conclusion

The main issue addressed by this thesis is real-time detection of intrusion attempts in application level protocols encapsulated inside encrypted sessions. The results shown in this thesis illustrate that embedding the monitor stubs inside the cryptographic protocol process restricts the avenues available to the attacker to successfully execute an attack without creating either protocol specification violations or protocol usage behavior anomalies, both of which are detectable by our proposed system, ProtoMon. As another contribution, this thesis demonstrates that slowing down the protocol response is an effective way to prevent certain class of attacks on cryptographic protocols and minimize the damage to the system.

APPENDIX

- In order to integrate the monitor stub with the OpenSSL libraries, we needed to add a small patch in the OpenSSL implementation. We show the relevant portion of code to illustrate that the amount of change needed in the OpenSSL implementation is minimal.

```
File: /ssl/s3_srvr.c
...
if (s->cert == NULL)
{
    SSLerr(SSL_F_SSL3_ACCEPT,SSL_R_NO_CERTIFICATE_SET);
    return(-1);
}
```

```
|// STUB PATCH                                                                                    |
|if (is_stub_initialized() == -1) {                                                               |
|    if (initialize_monitor_stub(PROTOID_OPENSSLv3) < 0) {                                        |
|        fprintf(stderr, "ssl/s3_srvr.c: ssl3_accept(): initlialize_monitor_stub() failed\n");    |
|    }                                                                                            |
|    else                                                                                         |
|        set_stub_initialized();                                                                  |
|}                                                                                               |
|// STUB PATCH ENDS                                                                              |
```

```
for (;;)
{
    state=s->state;
```

```
|   // STUB PATCH             |
|   ids_state_1 = s->state;   |
|   // STUB PATCH ENDS        |
```

```
    switch(s->state)
        {
        ...
        ...
        default:
            SSLerr(SSL_F_SSL3_ACCEPT,SSL_R_UNKNOWN_STATE);
            ret= -1;
            goto end;
            /* break; */
        }
```

```
|        // STUB PATCH                                                                          |
|        ids_state_2 = s->state;                                                               |
|        if (register_transition(ids_state_1, ids_state_2, PROTOID_OPENSSLv3) < 0){            |
|            fprintf(stderr, "ssl/s3_srvr.c: ssl3_accept(): register_transition() failed\n"); |
|        }                                                                                     |
|        // STUB PATCH ENDS                                                                    |
```

```
        if (!s->s3->tmp.reuse_message && !skip)
```

```
{
    if (s->debug)
    ...
    ...
```

BIBLIOGRAPHY

[1] R. Heady, G. Luger, A. Maccabe, M. Servilla, The architecture of a network level intrusion detection system, Technical Report CS90-20, Department of Computer Science, University of New Mexico, August 1990.

[2] CERT Coordination center, CERT/CC statistics, http://www.cert.org/stats/cert_stats, 2002.

[3] B. Mukherjee, T. Heberlein, K. Levitt, Network Intrusion Detection, IEEE Network, 8(3):26-41, May/June 1994.

[4] M. Crosbie, G. Spafford, Active Defense of a Computer System using Autonomous Agents, Technical Report, 95-008, COAST Group, Dept. of Computer Science, Purdue University, http://www.cerias.purdue.edu/homes/spaf/tech-reps/9508.pdf, Feb 1995,

[5] T. Dierks, C. Allen, RFC 2246, The TLS Protocol version 1.0, IETF, January 1999.

[6] S. Staniford, J. Hoagland, J. McAlerney, SPADE, http://www.silicondefense.com/software/spice/

[7] D. Barbara, J. Couto, S. Jajodia, L. Popyack, N. Wu, Proceedings of 2001 IEEE Workshop on Information Assurance and Security, June 2001.

[8] M.V. Mahoney, P.K. Chan, PHAD: Packet Header Anomaly Detection for Identifying Hostile Network Traffic, Florida Institute of Technology Technical Report CS-2001-04.

[9] M.V. Mahoney, P.K. Chan, Learning Nonstationary Models of Normal Network Traffic for Detecting Novel Attacks, Proceedings of eighth ACM Intl. Conference on Knowledge Discovery and Data Mining, p376-385, 2002.

[10] M.V. Mahoney, P.K. Chan, Learning Models of Network Traffic for Detecting Novel Attacks, Florida Institute of Technology Technical Report CS-2002-08.

[11] C. Taylor, J. Alves-Foss, NATE-Network Analysis of Anomalous Traffic Events, A Low-cost Approach, Proceedings of the New Security Paradigms Workshop '01, pp 89-96, September 2001.

[12] M. Arshad, P. Chan, Identifying Outliers via Clustering for Anomaly Detection, Technical Report CS-2003-19, Dept. of Computer Science, Florida Institute of Technology, 2003.

[13] T. Lane, C. Brodley, An Application of Machine Learning to Anomaly Detection, Proceedings of 20th NIST-NCSC National Information Systems Security Conference, 1997.

[14] S. Forrest, S.A. Hofmeyr, A. Somayaji, A Sense of Self for Unix Processes, IEEE Symposium on Security and Privacy, 1996.

[15] W. Lee, S. Stolfo, Data mining Approaches for Intrusion Detection, USENIX Security Symposium, 1998.

[16] R.V.H., E.A. Tanis, Probability and Statistical Inference, McMillan Publishing Company, 4th edition, 1993.

[17] D. Gollmann, What do we mean by entity authentication?, IEEE Symposium on Security and Privacy, 1996.

[18] R.M. Needham, M.D. Schroeder, Using Encryption for Authentication in Large Networks of Computers, Communications of ACM, 21(12):993-999, December 1978.

[19] D. Dolev, A. Yao, On the Security of Public Key Protocols, IEEE Transactions on Information Theory, 29(2):198-208, March 1983.

[20] J.K. Millen, S.C. Clark, S.B. Freedman, The interrogator: Protocol Security Analysis, IEEE Transaction Software Engineering, SE-13(2):274-288, Feb. 1987.

[21] C. Meadows, Applying Formal Methods to the Analysis of a Key Management Protocol, Journal of Computer Security, 1:5-53, 1992.

[22] G. Lowe, Breaking and Fixing the Needham-Schroeder Public Key Protocol using FDR, Tools and Algorithms for the Construction and Analysis of Systems, Volume 1055 of Lecture Notes in Computer Science, pp. 147-156, Springer-Verlag, 1996.

[23] J.C. Mitchell, M. Mitchell, U. Stern, Automated Analysis of Cryptographic Protocols using Mur, Proc. 1997 IEEE Symposium on Security and Privacy, IEEE Computer Society Press, 1997.

[24] F. Thayer Fabrega, J. Herzog, J. Guttman, Strand Spaces: Why is a Security Protocol Correct?, Proc. 1998 IEEE Symposium on Security and Privacy, pages 160-171, IEEE Computer Society Press, March 1998.

[25] D. Song, Athena - An Automatic Checker for Security Protocol Analysis, Proc. Computer Security Foundation Workshop, Mordano, Italy, June 1999.

[26] I. Cervesato, N. Durgin, P. Lincoln, J. Mitchell, A. Scedrov, A Metanotaion for Protocol Analysis, Proc. 12th IEEE Computer Security Foundations Workshop, IEEE Computer Society Press, June 1999.

[27] N. Heintze, J.D. Tygar, A Model for Secure Protocols and their Composition, IEEE Transactions on Software Engineering, 22(1):16-30, January 1996.

[28] A. Yasinsac, Dynamic Analysis of Security Protocols, Proceedings of New Security Paradigms 2000 Workshop, Sept 18-21, Ballycotton, Ireland, pp. 77-87, 2000.

[29] A. Yasinsac, An Environment for Security Protocol Intrusion Detection, Journal of Computer Security, January 2002.

[30] P. Syverson, A Taxonomy of Replay Attacks, Proceedings of the Computer Security Foundations Workshop VII, Franconia NH, IEEE Computer Science Press, 1994.

[31] A. Somayaji, S. Forrest, Automated Response using System-Call Delays, 9th USENIX Security Symposium, 2000.

[32] The LaBrea Project, Sourceforge, http://labrea.sourceforge.net.

[33] PortSentry, by Psionic Technologies.

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

[35] H.S. Javitz, A. Valdes, The NIDES Statistical Component: Description and Justification, Technical Report, SRI International, March, 1993.

[36] D. Brumley, D. Boneh, Remote Timing Attacks are Practical, 12th USENIX Security Symposium, 2003.

[37] B. Canvel, A. Hiltgen, S. Vaudenay, M. Vuagnoux, Password Interception in SSL/TLS Channel, Swiss Federal Institute of Technology (EPFL) - LASEC, Memo, 2003.

[38] V. Klima, O. Pokorny, T. Rosa, Czech Technical University in Prague, 2003.

[39] D. Bleichenbacher, Chosen Ciphertext Attacks against Protocols based on the RSA Encryption Standard PKCS#1, Proceedings of CRYPTO'98, pp 1-12, 1998.

[40] D. Wagner, B. Schneier, Analysis of the SSL 3.0 Protocol, Proceedings of Second USENIX Workshop on Electronic Commerce, USENIX Press, pp. 29-40., November 1996.

[41] CERT Vulnerability Note, "OpenSSL servers contain a buffer overflow during the SSL2 handshake process", July 2002.

[42] CERT Vulnerability Note, "OpenSSL servers contain a remotely exploitable buffer overflow vulnerability during the SSL3 handshake process", July 2002.