The International Journal Of Engineering And Science (IJES)
|| Volume || 3 || Issue || 3 || Pages || 30-36 || 2014 ||
ISSN (e): 2319 – 1813 ISSN (p): 2319 – 1805

A Knowledge-Based Intrusion Detection Engine to detect attacks on security protocols

[1,] Asst. Prof. Pratima Bhalekar & [2,] Asst. Prof. Sonali Ingle
[1, 2,] *Ashoka Center for Business and Computer Studies,*
Wadala, Nashik

--ABSTRACT--
With the evolvement of the Internet over the last few years, the need for security has been rising with it mainly due to the openness and connectivity nature of the web, people and organizations are faced with more challenges every day to secure their data and all other assets of value to them. No system is totally secure. Any security procedures should be undertaken with that in mind. There will always be threats and actual intrusions. The ultimate goal should be minimizing the risk and not eliminating it. This paper describes a system for detecting intrusions, introducing technologies to provide protection for electronic information exchange over public networks.

KEYWORDS: *Intrusion Detection, Security Protocol, Attacks, IDE.*

Date of Submission: 28 February 2014 ⟺ Date of Acceptance: 15 March 2014

I. INTRODUCTION

Network security refers to any activities designed to protect your network. Specifically, these activities protect the usability, reliability, integrity, and safety of your network and data. Effective network security targets a variety of threats and stops them from entering or spreading on your network. Our research combines two common security technologies to provide protection for
electronic information exchange over public networks.

1.1 Background

Current technology for computer and data security is usually based upon Access Control List (ACL) methodology, monitored environments, or data encryption. The use of encryption grew dramatically after introduction of the Data Encryption Standard [1] and public key technology [2], both in the late 70s. In this paper, we demonstrate a new security technique based on monitoring encrypted exchanges in order to detect intrusions.

1.1.1. Intrusion Detection

Intrusion detection (ID) is a type of security management system for computers and networks. An ID system gathers and analyzes information from various areas within a computer or a network to identify possible security breaches, which include both intrusions (attacks from outside the organization) and misuse (attacks from within the organization). ID uses *vulnerability assessment* (sometimes refered to as *scanning*), which is a technology developed to assess the security of a computer system or network.

Intrusion detection functions include:
- Monitoring and analyzing both user and system activities
- Analyzing system configurations and vulnerabilities
- Assessing system and file integrity
- Ability to recognize patterns typical of attacks
- Analysis of abnormal activity patterns
- Tracking user policy violations

There are two main designs available to IDSs for detecting attacks: 1) the misuse detection design and 2) the anomaly detection design [2].

A strength of misuse detection paradigm is that when it signals that an attack has occurred, it is very likely that an attack has actually occurred. In IDS terminology, it minimizes false positives. A weakness of misuse detection is that only attacks recorded in the database can be recognized. New attacks (and other attacks that have not yet been entered in the database) cannot be recognized. This results in failure to report some attacks (termed "false negative"). The behavior-based design uses statistical methods or artificial intelligence in order to detect attacks. The strength of anomaly detection systems is that they can detect new attacks and there is no requirement to enter attack signatures into a database.Conversely, anomaly detection systems have a higher false alarm (or false positive) rate, because they sometimes report different, but non-malicious, activity as an attack. The current and continuous reports of newly discovered flaws and vulnerabilities in end-user and architectural systems indicates that we will likely never be able to guarantee the security of electronically transmitted information. Moreover, it strongly suggests that preventative methods will likely never be sufficient to protect our networks. Our approach combines complementary prevention (encryption) and detection (IDS) technologies to provide layered security for network traffic.

1.1.2.Security Protocols

A **security protocol** (**cryptographic protocol** or **encryption protocol**) is an abstract or concrete protocol that performs a security-related function and applies cryptographic methods. Algorithms such as DES, the International Data Encryption Algorithm and the Advanced Encryption Standard make use of keys to encrypt plain text messages before they are transmitted. Security protocols allow key exchange, authentication, and privacy through strong encryption. These protocols define the content and order of exchanges between the communicating principals. Early security protocols were short, usually with less than five messages. They were also simple, often developed for execution in a single, non-current session, with no branching or decision mechanisms. The classic Needham and Schroeder Conventional
Key Protocol is representative of early protocols and is shown in Figure 1.

A □ □ S : A, B, Na
S □ □ A : E(Kas: Na, B, Kab, E(Kbs : Kab, A))
A □ □ B : E(Kbs : Kab, A)
B □ □ A : E(Kab : Nb)
A □ □ B : E(Kab : Nb– 1)

Figure 1

Unfortunately, encryption backed by carefully crafted and thoroughly tested security protocols may still not be sufficient to prevent sophisticated intruders from compromising secure communication. Subtle flaws exist in many security protocols that can be used by malicious parties to compromise the security goals by subverting the underlying protocol. For example, sophisticated intruders may be able to spoof valid parties in a data exchange by using replay techniques where information from previous runs of any encrypted exchanges are used in the current run, as shown by Denning and Sacco [3]. As a result, intruders may be able to masquerade as valid parties in communication, steal keys etc. which leads to compromise of the encrypted exchange. It is clear that another level of protection must be provided for encrypted data exchanges to detect attacks on the security protocols.

1.1.3. The Secure Enclave Attack Detection System

The Secure Enclave Attack Detection System (SEADS) [6] is a system that can detect attacks on security protocols within an enclave of valid and recognized parties that communicate using a public network. In this environment, security protocol activity based on the message exchanges within the enclave is gathered by an Activity Monitor and compared against a knowledge base of attack signatures on protocols. This allows the Intrusion Detection Engine (IDE) to detect attempts to subvert the security protocols and to identify suspicious activities. The SEADS architecture is shown in Figure 2.

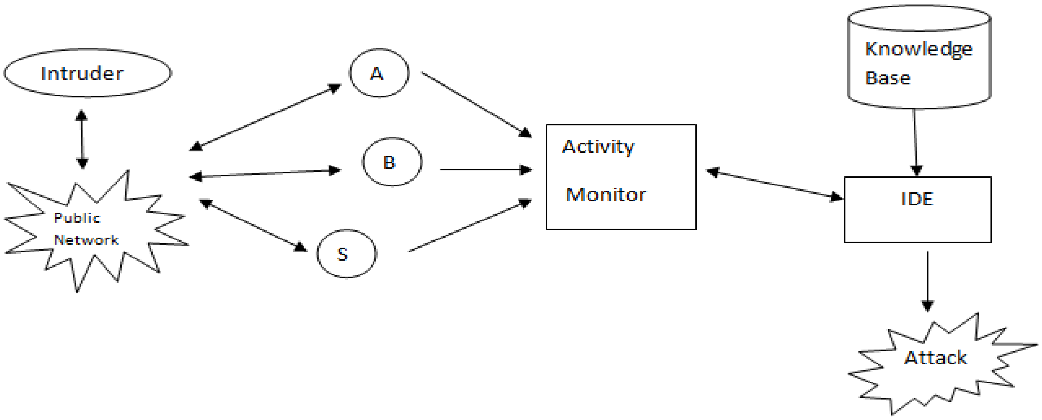

Figure 2.

The detection mechanism of the Intrusion Detection Engine (IDE) is constructed based on the knowledge-based paradigm. The IDE detects anomalous, malicious, or suspicious protocol activity occurring within the secure enclave based upon previously gathered attack signatures. We know of no other executing environment that can detect attacks against encrypted traffic.

II. DETECTING INTRUSIONS USING SECURITY PROTOCOL CHARACTERISTICS

The goal of our research is to show that formal definitions of attacks on security protocols can be represented as signatures that can be stored in a knowledge base and compared against ongoing activity to detect attacks. This is done using specific characteristics of protocols. When our system recognizes a specific signature of activity that corresponds to a known attack, we signal that an attack has occurred. Additionally, because of the characteristics of our system, we are also able to identify suspicious behavior that may or may not represent an attack. Again, this suspicious activity is recognized based on the characteristics of the security protocols that we monitor, not on the longterm behavior of any principal(s). From this perspective, our technique may be considered *online* analysis of Security Protocols. Moreover, we know of no other project that analyzes executing security protocols. Moreover, our environment consists of a real world scenario comprised of multiple users engaged in multiple concurrent sessions, and using many different protocols, with all the traffic interleaved.Finally, we also define and utilize signatures of properly executing protocols as part of our detection paradigm.

2.1. Constructing Signatures of Attacks

An important feature of the our technique is that the detection mechanism does not rely upon knowledge of the payload of the messages exchanged between the principals during protocol sessions. This is because the IDE detects attacks based upon the characteristics of the security protocols themselves. The signatures constructed from protocols and their known attacks are represented by:
(1) The protocols that are in use
(2) The principals (originator and recipient) involved
(3) The messages that are sent
(4) The messages that are received
(5) The concurrent sessions that occur

Consider the canonical *Needham and Schroeder Conventional (symmetric) Key Protocol* (NSCKP) [5] shown in Figure 1. This protocol requires three principals: A, B and the trusted third party server S. The aim of NSCKP is to establish a secret key *Kab* that is to be used by the principals A and B to encrypt their future exchanges. At the end of a correct run of the protocol, both principals should be in possession of the secret key, Kab, newly generated by the server S.The given description of the protocol includes information about the payload data exchanged by the principals. However, as previously mentioned, the IDE does not rely on payload information for its detection mechanism. Rather, it relies on the proper sequencing of messages in the session. The NSCKP can be represented by the signature given in Figure 3

Protocol	Session	Message #	Action	Sender	Receiver
NSCKP	x	1	send	A	S
NSCKP	x	1	receive	A	S
NSCKP	x	2	send	S	A
NSCKP	x	2	receive	S	A
NSCKP	x	3	send	A	B
NSCKP	x	3	receive	A	B
NSCKP	x	4	send	B	A
NSCKP	x	4	receive	B	A
NSCKP	x	5	send	A	B
NSCKP	x	5	receive	A	B

Figure 3

Each step of the signature is considered an *event*. Alice1 sending a message to Sally is considered as a 'send' event and similarly Sally receiving a message from Alice is a 'receive' event by Sally from Alice. An important feature of protocol signatures is that they include receive events. Earlier research [6] took into account only the message sending events in the protocol signature. This means that Alice sending a message to Sally (as in event 1), and correspondingly Sally receiving the same message (event 2) will be represented as two distinct events in the protocol signature used by the IDE.

Consider a scenario during the run of the NSCKP. Upon sending a message to Sally as part of the first step of the protocol, Alice will inform the activity monitor of SEADS about this.
Since a public network is being used for the message transfer between Alice and Sally on insecure lines, the message may be lost or may be intercepted by an intruder. In either case Sally will not inform the monitor that it actually received a message from Alice. Thus, the sequence of events logged in the monitor will show a message sent by Alice to Sally, but not received by Sally, as evident by the lack of the receive notification by Sally to the monitor. It is prudent therefore to include the message receipt as a separate event in the protocol signature, as we further illustrate.

The attack on the Needham and Schroeder Conventional Key Protocol was demonstrated by Denning and Sacco [4]. The attack leverages the lack of temporal information in message three. Although Bob decrypts this message and legitimately assumes that it was created by the server Sally, there is nothing in the message to indicate that it was actually created as part of the current protocol run. Thus, suppose, a previously distributed key Kab has been compromised, through cryptanalysis or other means, and is known by a malicious intruder, Mallory. If Mallory monitored and recorded message three of the corresponding protocol run, consisting of E(Kbs: Kab, A), he can now fool Bob into accepting the key as new by the protocol given in Figure 4.

(3) *M(A) □ □ B : E(Kbs : Kab, A)
(4) B □ □ M(A) : E(Kab : Nb)
(5) M (A) □ □ B : E(Kab : Nb–1)
*M (A) stands for *M* masquerading as A.

Figure 4

After effecting the attack, Bob believes he is following the correct protocol. Mallory is able to form the correct response in (5) because she knows the compromised key Kab. She can now engage in a communication with Bob using the compromised key and masquerade as Alice.
We can generate a signature recognizable by the IDE for the above attack on the Needham and Schroeder protocol. The signature is comprised of only three events, two receive events and a send event as shown in Figure 5.

Protocol	Session	Message #	Action	Sender	Receiver
NSCKP	x	3	receive	A	B
NSCKP	x	4	send	B	A
NSCKP	x	5	receive	A	B

Figure 5

Since the malicious intruder (M), is not part of the secure enclave, it will not co-operate with the activity monitor and, hence, will not inform the monitor whenever it sends or receives messages. Thus the above attack signature will consist only of events reported by Bob (a valid principal) to the monitor.

2.2. The Recognition Machine

In section 2.1 we described in detail how the attack signatures are constructed from the description of security protocols. The IDE interfaces with the activity monitor to receive events corresponding to protocol sessions executing within the enclave and compares the events with the attack signatures stored in the knowledge base. The comparison mechanism in the IDE is achieved by using Finite State Machines.

2.3. Signature Format in the Knowledge Base

Each signature is stored in the Knowledge Base as a procedure defining a finite state machine. Information in the first line identifies the entry, followed by the state identifiers and the transitions that occur.

2.4. Construction of the Finite State Machine

When a session begins, the IDE constructs a Finite State Machine (FSM) recognizer for each signature stored in the knowledge base, corresponding to the protocol used in that session. The state transition diagram for attack signature #1 on the NSCKP protocol (as described in section 2.1) is shown in Table 1.

Initially the recognizer will be in the start state (SS). As the IDE receives events from the monitor for this particular protocol session it advances the FSM for this signature if the arriving events match those in the attack signature. Upon a transition to the final state in any of the finite state machines corresponding to the attack signatures of the protocol, the IDE signals an attack notification.

Current State	Event	Protocol	Session	Sender	Receiver	Message Number	Next State
SS	receive	NCCKP	X	A	B	3	S1
S1	send	NCCKP	X	B	A	4	S2
S2	receive	NCCKP	X	B	A	5	FS

Table 1

2.5 How to Detect Protocol Attack

IDE uses different methods to detect protocol attack depending on number of sessions used in specific attack. Attacks on security protocols may be over only a single session of the protocol or may utilize information gleaned from multiple runs of the protocol. Thus, attacks may be classified as *Single session* attacks or *Multi-session* attacks.

2.6. Single Session Attacks

Single session attacks are those attacks which may occur in a single session. The signature of such an attack may differ from the protocol itself in only something so subtle as a missing receive statement.

Detection of single session attacks by the IDE is simply a matter of the relevant attack finite state machine reaching the final state, upon which the IDE will signal a notification. No knowledge of the previous session is necessary for the IDE to detect this attack.

2.7. Multi-Session Attacks

Multi-session attacks are those attacks that use information extracted from more than one previous or concurrent protocol sessions. For multi-session attacks, the IDE classifies them as either Replay Attacks or Parallel Session Attacks.

2.7.1 Replay Attacks

Also known as a "man-in-the-middle attack". A **replay attack** is a form of network attack in which a valid data transmission is maliciously or fraudulently repeated or delayed. This is carried out either by the originator or by an adversary who intercepts the data and retransmits it. The first question that must be answered is: "How much time can pass between the reference session and the attack session?" This is an important question in our architecture because of the way replay attacks are detected. The signature of a replay attack consists of the signature of the reference session followed by the signature of the attack session. Thus, the recognizer must remain active until either an attack is detected or the threshold period expires.

We handle this by requiring the author of signatures of replay attacks to include the threshold in the signature, which will vary from protocol to protocol. The default wait constant was chosen to be ten seconds for the IDE prototype. If events occur that triggers a replay recognizer, if the time difference between the attack session and the reference session is greater than the wait time, the IDE will flag this activity as suspicious behavior.

2.7.2 Parallel Session Attacks

A parallel session attack occurs when two or more protocol runs are executed concurrently and messages from one run (the reference session) are used to form spoofed messages in another run (the attack session).

To initiate the attack, Mallory waits for Alice to initiate the first protocol session with Bob.Mallory intercepts the message and pretends to be Bob, starting the second run of the protocol by replaying the intercepted message. Alice replies to Mallory's challenge with exactly the value that Mallory requires to accurately complete the attack session. The attack is shown in Figure 7.

Attack Session	Reference Session
A □ □ M(B): E(Kab : Na)	
	M(B) □ □ A: E(Kab : Na + 1)
	M(B) □ □ A: E(Kab : Na)
A □ □ M(B): E(Kab :Na + 1)	

Figure 7

The IDE detects parallel session attacks by matching the ongoing activity against the attack signatures. The telling factor in this case is the omission of any information from Alice's partners in either session, as reflected in the signature in Table 2.

Current State	Event	Protocol	Session	Sender	Receiver	Message Number	Next State
SS	Send	OWAP	X	A	B	1	S1
S1	Receive	OWAP	X+α	B	A	1	S2
S2	Send	OWAP	X+α	A	B	2	S3
S3	Receive	OWAP	X	B	A	2	FS

Table 2

III. DESIGN OF THE INTRUSION DETECTION ENGINE

The design of the IDE uses the objectoriented paradigm. The problem was broken down into smaller components, and appropriate classes were developed to accurately represent the problem.

A major factor in the design of the IDE, was the complexity of the environment being monitored. Within any enclave, we expect to monitor events interleaved from multiple:

□ □ □ Concurrent sessions
□ □ □ Different principals
□ □ □ Different protocols

In addition there is no guarantee that all the sessions will properly conclude. Some sessions may be suspended abnormally and messages may be lost.

3.1. Architectural Design

A number of issues had to be taken into account in the design phase of this research implementation. The design was created in order to ensure that all the requirements and specifications were satisfied.In the secure enclave it is possible to have multiple concurrent sessions of different protocols executing within the enclave. The sessions may consist of the same or different principals. The Intrusion detection engine must be able to keep track of the different protocol sessions executing within the enclave in order to detect any attacks or suspicious activity. Not all attacks on security protocols occur over a single session. As described earlier, multi-session attacks such as replay attacks or parallel attacks may occur within the enclave. These multi-session attacks span multiple different protocol sessions. The Intrusion detection engine must provide a means to keep track of such executing sessions and detect any attacks.

Additionally, the detection of attacks has to be communicated to the person or system monitoring the enclave. Detailed reports of all attacks or suspicious behavior must be generated by the IDE. Such reports provide in-depth information about the type of attack and principals participating in the protocol session. The Intrusion Detection Engine receives crucial inputs from the Activity Monitor and from the Knowledge base of protocol signatures. It is important to ensure that interfaces with the Monitor and the Knowledge base are well-defined and reliable.

3.1.1 The Thread Dispatcher and Monitors

As noted earlier, the IDE receives protocol events from the monitor as they occur. The IDE is multi-threaded with a single thread to serve as the thread dispatcher. Since each protocol may have many attack signatures associated with it, when a new protocol session begins, the IDE spawns a new thread to monitor all the FSM recognizers for that protocol. As illustrated in Figure 8, the Thread Dispatcher then routes events to the appropriate thread as they arrive.
To keep track of all the threads existing within the system, a *ThreadList* class is employed, that holds the protocol name, session number, identifiers of the principals involved, a signal to which the thread listens, and a thread identifier for each thread.

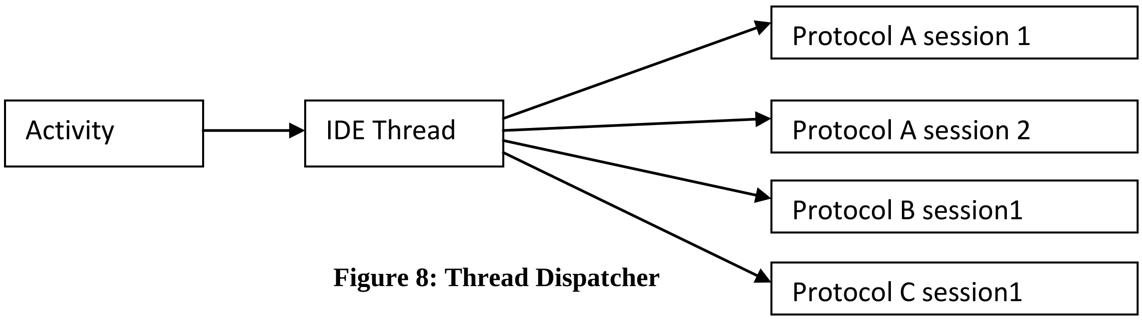

Figure 8: Thread Dispatcher

IV. THE GRAPHICAL USER INTERFACE

In our research, a Graphical User Interface (GUI) was implemented for an overall view of the attacks and suspicious activities detected within the enclave. The GUI allows the reporting of attacks to the user. The user can specify the time duration and the protocol name to obtain a detailed report of all the attacks (on the specific protocol) that took place during that period.

V. CONCLUSION

We have designed and implemented a Knowledge-Based Intrusion Detection Engine to detect attacks on security protocols executing within a secure enclave. This research provides an necessary extra level of protection for encrypted exchanges.Extensive research on the characteristics of security protocols enabled this detection methodology to achieve its desired functionality. Extracting the description of security protocols into sequences of events allows the IDE to detect attacks on those protocols. The IDE will detect any attacks or suspicious activity on security protocols executed by valid principals operating within a secure enclave. The detection of the IDE compares protocol activity gathered by the Monitor against the attack signatures stored in the Knowledge base. A Graphical User Interface (GUI) was also developed in order to facilitate an overall report of attacks that have been detected by the IDE, along with their occurrence times. Collectively, these components represent a fully functional Secure Enclave Attack Detection System.

REFERENCES

[1] John Clark & Jeremy Jacob, "Attacking Authentication Protocols", High Integrity Systems 1(5):465-474, August 1996.
[2] H.Debar, M.Dacier, A.Wespi, "Towards a Taxonomy of Intrusion Detection Systems", Elsevier Science B.V 31 (1999) 805-822
[3] Dorothy E. Denning, "An Intrusion-Detection Model", From 1986 IEEE computer Society Symposium on Research in Security and Privacy.
[4] Dorothy Denning and G.Sacco, "Timestamps in Key Distribution Protocols", Communications of the ACM, 24(8), August 1981, pp. 533-534.
[5] Roger M. Needham and Michael Schroeder, "Using Encryption for Authentication in Large Networks of Computers", Communications of the ACM, 21(12), Dec. 1978, pp. 994-995.
[6] Alec Yasinsac, "Detecting Intrusions in Security Protocols", Proceedings of First Workshop on Intrusion Detection Systems, in the 7th ACM Conference on Computer and communications Security, June 2000, pp. 5-8.
[7] National Bureau of Standards (NBS). Data Encryption Standard. Dederal Infor-mation Processing Standard, Publication 46, NBS, Washington, D.C., January 1977
[8] R.L Rivest, A. Shamir, L. M. Adleman, "A Method for Obtaining Digital Signatures and Public Key Cryptosystems", CACM, Vol. 21, No. 2, Feb 1978, pp. 120-126
[9] Otwy, D., and Rees, O. 'Efficient and timely mutual authentication'. Operating Systems Review 21, 1(Jan. 1987), pp. 8-10
[10] J. Kelsey, B. Schneier, & D. Wagner, "Protocol Interactions and the Chosen Protocol Attack", Sec Protocols, 5th, Internat Wkshp Apr. 97, Proc. Springer-Verlag, 98, pp. 91-104