This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the ICC 2007 proceedings.

Combating Against Attacks on Encrypted Protocols

Zubair Md. Fadlullah[1,*], Tarik Taleb[1,∓], Nirwan Ansari[2], Kazuo Hashimoto[1], Yutake Miyake[3], Yoshiaki Nemoto[1], and Nei Kato[1].

[1]Tohoku University, Japan
[2]New Jersey Institute of Technology, USA
[3]KDDI Labs, Japan

Emails:
*zubair@it.ecei.tohoku.ac.jp
∓talebtarik@ieee.org

***Abstract*— Attacks against encrypted protocols are becoming increasingly popular. They pose a serious challenge to the conventional Intrusion Detection Systems (IDSs) which heavily rely on inspecting the network packet fields and are consequently unable to monitor encrypted sessions. IDSs can be broadly categorized into two types: signature-based and anomaly-based IDSs. The signature-based IDSs rely on previous attack signatures but are often ineffective against new attacks. On the other hand, anomaly-based detection systems depend on detecting the change in the protocol behavior caused by an attack. The latter can be employed to detect novel attacks, and therefore are often preferred over their signature-based counterpart. In this paper, we envision an anomaly-based IDS which can detect attacks against popular encrypted protocols, such as SSH and SSL. The proposed system creates a normal behavior profile and uses non-parametric Cusum algorithm to detect deviation from the normal profile. Upon detecting an anomaly, the proposed mechanism generates an alert, sets a delay to the protocol response, and traces back the attacker. The effectiveness of the proposed detection scheme is verified via simulations.**

I. Introduction

The ever growing use of network-based applications demands security against malicious attacks such as IP spoofing, Denial of Service (DoS), and Man-in-the-middle attack (MITM) through the use of encrypted protocols. Most modern servers accommodate cryptographic or encrypted protocols such as Secure Socket Layer (SSL), Transport Layer Security (TLS) and Secure Shell (SSH) protocols, which provide means of encrypting the otherwise visible "plain-text" from the application layer and nicely sit on top of transport layer protocols such as Transport Control Protocol (TCP). SSL/TLS are used to provide client-server authentication to establish a secure session by a mutual "handshake" through which the client and server can communicate both ways in a secure manner. On the other hand, SSH provides secure communications, often using username/password based authentication between the client and the server. These encrypted protocols are, however, subject to exploits, unauthorized accesses, and attacks.

The main motivation behind this paper comes from realizing the fact that attacks against encrypted protocols do exist in practice. Additionally, more sophisticated attacks against cryptographic protocols are emerging. A prime example of such attacks is the remote timing attack [1] against OpenSSL server which, under typical conditions, is able to extract the private key used by the server in just over six hours.

This work was inspired by the existing anomaly-based IDS called "Protomon" [2]. Although Protomon offers a unique approach to learn the normal behavior profile of the network and provides specification-based anomaly detection with a lower false alarm rate, it has some significant shortcomings like the use of a simple threshold-based mechanism to set up alarms and lack of an effective damage control scheme. Our paper addresses these issues efficiently and devises adequate solutions. Moreover, the scope of our work is not necessarily limited to only detecting attacks against encrypted protocols. The proposed approach monitors unencrypted portion of the TCP header and extracts features which can be used for detecting anomalies. Certain modifications of this approach may also be used for detecting some attacks against non-encrypted protocols. In this work, we use the non-parametric Cusum algorithm [3], a time series statistical model to determine the threshold depending on the network conditions based on which anomalies are detected. To devise this model, a database is designed to store snapshots of essential network and protocol parameters under normal network conditions (i.e., no attacks). The non-parametric Cusum algorithm can detect subtle-most changes in the network behavior that should be able to warn against even a slight malicious activity. We then attempt to evaluate the performance of this approach by measuring the latency between the attack-instance and attack-detection for different parameters. Finally, in the proposed scheme, the network topology consists of monitoring stubs placed in a distributed fashion, which seamlessly cooperate among one another. This cooperative strategy is used to perform traceback.

The remainder of this paper is organized as follows. Section II surveys some related works on intrusion detection systems. Section III describes the network topology of the proposed architecture. The section also presents the scope of attacks detectable by the proposed method and designs the database which provides inputs to the non-parametric Cusum algorithm used as the anomaly detector in the proposed scheme. The performance of the proposed scheme is evaluated in Section IV. Section V concludes the paper.

II. Related work

Over the recent years, Intrusion Detection Systems (IDSs) have been the primary tool in detecting malicious activities or attacks. Most IDSs investigate the network packet headers and provide effective detection of attacks against application protocols. However, these conventional IDSs are often unable to detect attacks against protocols which use encrypted sessions.

Primarily there are two approaches to detect intrusions, namely signature-based and anomaly-based detection techniques. The signature-based or rule-based detection technique

1-4244-0353-7/07/$25.00 ©2007 IEEE

uses a database of previously known attacks to compare and match an attack. Though this technique is effective in detecting a known attack, it fails to detect an attack for which there is no signature stored. On the contrary, anomaly-based IDSs are used to first learn the normal profile. At the advent of an attack, an anomaly-based technique detects a notable deviation from the normal profile which signals a potential attack. Even in case of a novel attack that disturbs the normal state of the network learnt during the profiling stage, the attack can still be detected. However, anomaly-based intrusion detection techniques are prone to higher false alarm rates.

Recent researches have focused on detecting various attacks against cryptographic protocols such as SSL/TLS and SSH. OpenSSL, the widely used implementation of SSL, is particularly vulnerable to many attacks such as timing attacks [1] where the attacker measures the time an OpenSSL server takes to respond to decryption queries. Consequently, the attacker is able to extract the private key stored in the server. Canvel $et\ al.$ [4] illustrate password attacks against Internet Messages Access Protocol (IMAP) servers using SSL-tunnel with its peer users. SSL version 3 is also susceptible to Version Rollback attack [5] where an attacker forces the SSL server to downgrade its SSL version. The attacker then exploits the vulnerabilities of the lower version of SSL. There have been numerous buffer overflow attacks against openSSL servers causing DoS. For instance, Slapper worm [6] propagated through the internet posing a serious denial of service threat to Apache webservers which use the MOD-SSL library.

The use of SSL/TLS session-aware user authentication [7] is a new concept that can thwart the MITM attack against SSL/TLS-based client/server communications. In this approach, a client first authenticates and provides some credential to the valid SSL/TLS-based server. A User Authentication Code (UAC) is generated based on this credential and the initial SSL/TLS session. Even if an attacker hijacks the connection and intercepts the UAC from the client, the attacker cannot modify the contents of UAC, since it is encrypted. To imposter as a client, the attacker has to use his or her own SSL session to communicate with the server. Retransmitting the intercepted UAC will be useless because the UAC is session specific. When the server sees that the submitted UAC has a different session ID than the current session ID, it will immediately suspect that the UAC came from some entity other than the valid client. The server will then drop this suspicious session.

WebSOS [8] is an overlay-based architecture that ensures client access to a webserver under a DoS attack. Traffic from a source to a target server uses a route through access points, beacons, and servlets. End-to-end communication between a client and the server is encrypted using a SSL session. If an access point is attacked, an alternate access point is chosen to enter the overlay. If a beacon is attacked, the node exits the overlay and new paths are discovered over the changed overlay topology.

Protomon [2] describes an anomaly-based IDS for cryptographic and application protocols with low false positive rates. Protomon operates in three modes, namely Learn, Detect and Prevent modes. In the Learn mode, a monitoring stub per server constructs normal usage patterns for the monitored protocol by considering various parameters such as the maximum number of protocol-sessions and maximum number of broken sessions during the sampling time. During the Detect mode, online observations are constantly compared with the acceptable threshold of normal profiles over long and short term distributions. When Protomon detects an anomaly, it goes to the Prevent mode. In the Prevent mode, the monitoring stub slows down the protocol response, forcing the anomaly to stay within the threshold. The delay is revoked when no more anomaly is detected.

Protomon offers a unique solution by integrating lightweight protocol monitoring processes with the protocol library. However, it uses a simple arbitrary threshold to determine the anomaly. The Protomon provides a mere damage control mechanism by imposing delay in the Prevent mode, and it does not provide any traceback mechanism. Furthermore, a monitoring agent for each server serves the purpose but perhaps not as efficiently as compared to a distributed set of monitoring stubs exchanging information. Our work presents a solution to these problems by using a dynamic thresholding scheme to detect anomaly, tracing back the roots of attackers, and distributing unique monitoring agents over the network topology, as will be described in the next section.

III. Proposed Architecture

A. Envisioned Attacks

There exist many attacks against cryptographic protocols. Examples of such attacks depend on the design and implementation of the protocol. For example, the OpenSSL implementation of SSL is vulnerable to specific remote timing attacks, MITM attacks, buffer overflow attacks [9] [10], and Version Rollback attacks. In the remote timing attack, SSL renegotiation attack, and password attack (also known as dictionary attack or brute force attack), there is a high interaction between the attacker and the cryptographic protocol server. In case of a buffer over-flow attack, few messages are exchanged between the client and the server. Let us call such attacks as 'low interactive attacks'. From this observation, we broadly categorize the different attacks into two types: highly interactive attacks and low interactive attacks. In order to detect a low interactive attack, it is sufficient to define a strict protocol state change directly at the server. Our target is to detect the highly interactive attacks.

B. Considered Network Topology

The envisioned network topology as shown in Fig-1 consists of a number of servers running services based on both encrypted and application level protocols. Users from an untrusted network or from the Internet can connect to any one of these servers. Our approach is not to implement the IDS at the servers, which would put additional load on the servers. Rather, the IDS is implemented aside of network elements such as routers. We call each IDS entity a monitoring stub

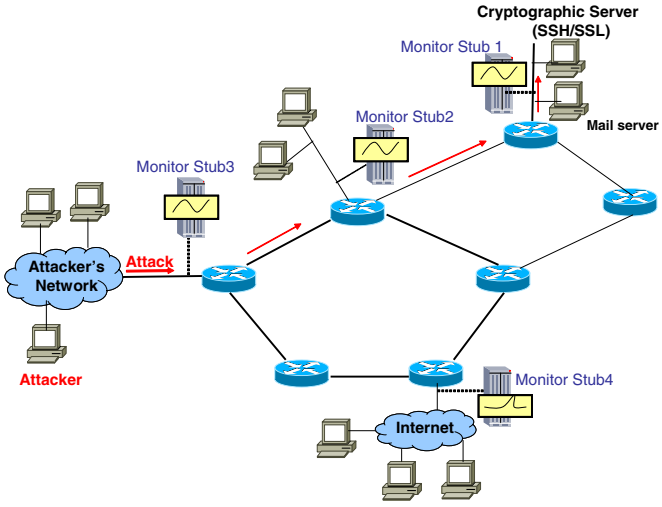


Fig. 1. Envisioned architecture.

(MS). Rather than placing a MS for each server as in [2], MSs are strategically distributed over the entire network. The functionality of a MS is described in the following subsection.

C. Functionality of a Monitoring Stub

Each MS is a packet sniffing entity implemented close to a router. For application level protocols, it is possible to sniff the network packet headers as well as the payloads, and inspect and analyze them at a later instant. For encrypted protocols, a MS uses the tcpdump utility to monitor the TCP headers which are not encrypted. For instance, detection of a failed SSH session by a MS requires the system to know how the SSH protocol works in the transport layer level. A client first attempts to establish a connection to the server by sending a SYN packet. The server acknowledges this by sending an ACK packet and a SYN packet of its own. If the client successfully logs onto the server and wants to quit, the client will initiate the FIN packet first. On the other hand, if the server initiates the FIN packet first, it means the server is closing the connection either due to an invalid "login" attempt or a time out. If a monitoring stub monitors that the server is the first originator of the FIN packet soon after the connection attempt, it deems the event as a "failed session".

To serve the purpose of an IDS, a MS performs a number of functions: learning normal profiles, monitoring, generating alerts, and tracing back the attacker.

Learning Phase

In order to detect anomalies, it is important to study the protocol implementations and standard documents such as Request for Comments (RFCs), from which we can define the normal mode of the protocol operations. Furthermore, the protocol behavior in a network is not likely to remain the same all the time. For example, protocol behavior during day-time may be different than that at night. So, a statistical profile over time is also developed in the profiling phase. To do so, profiling is done during the normal network conditions or near normal network conditions with low acceptable levels of anomaly. Each MS creates a database with carefully chosen features that are extracted from monitored traffic over time. These features are chosen and recorded in such a way that they can serve as parameters to the non-parametric Cusum algorithm used by the MS in the detection phase. The format of a typical table from the database is shown in Fig-2. S_n and T_n are the two features that are extracted for a target protocol (e.g., SSH, SSL) over the profiling interval, Δ_n. S_n and T_n indicate the number of failed sessions and number of total sessions, respectively. Using these two values, the fraction of failed sessions, F_n, is then computed and stored.

Detection Phase

We have chosen the non-parametric Cusum algorithm over the classical version because pre-change and post-change distributions for Internet-based applications including cryptographic applications are usually not known *a priori* [11]. Here, we analyze a random sequence consisting of the number of failed SSH sessions in a time interval. We obtain the fraction of failed sessions, F_n, in a time interval, Δ_n [n=1,2,3... and Δ_{n+1}=Δ_n]. The mean of F_n, m, over the profiling period of normal scenario is much less than one and remains close to zero until an anomaly occurs [Fig-3(a)]. However, one assumption of the non-parametric Cusum algorithm is that the mean value of the random sequence is negative during the normal conditions and becomes positive when a change occurs. So, it is necessary to transform F_n into a new sequence G_n. The transformation is given by (G_n = F_n - β) as shown in Fig-3(b), where β is the transformation parameter [11] such that ($\beta >> m$). We compute β by taking the average of the highest F_n values obtained during the profiling period. During an attack, the increase in the mean of G_n during an observation period compared to that during the profiling period can be lower-bounded by h, which is typically set as twice the value of β. The problem of the online detection of attacks is solved by employing a recursive version of the non-parametric Cusum algorithm defined by a new sequence Y_n as follows:

$$Y_n = (Y_{n-1} + G_n)^+; Y_0 = 0 \qquad (1)$$

where $x^+ = x$ if $x > 0$; otherwise, $x^+ = 0$.

A large value of Y_n is a strong indication of an anomaly, i.e., a potential attack. A constant θ, for this profiling, is adjusted as a threshold for detecting attacks as shown in Fig-3(c). θ is computed by the following Equation [12]:

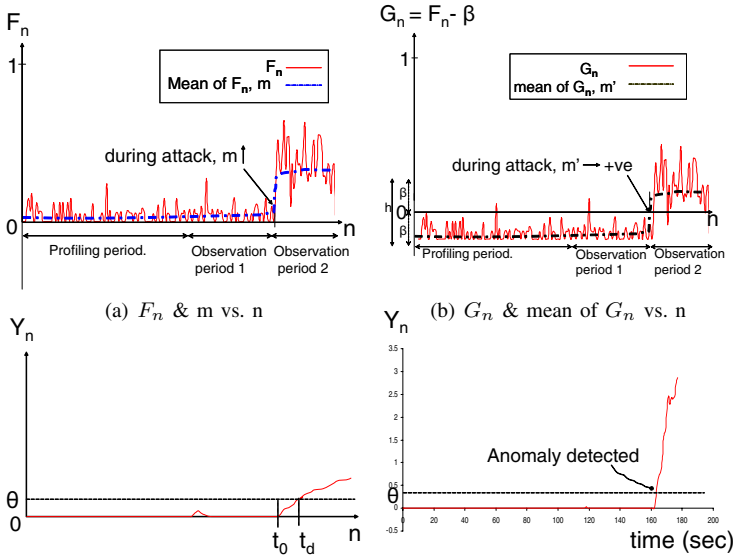
$$\theta \simeq [h\text{-}(\beta\text{-}m)] \cdot (t_d - t_0)^+ \qquad (2)$$

In the absence of attacks, the G_n values lie mostly lower bounded by (-β). During an attack, the G_n values become positive and substantially large. As a tolerable margin of this change, h is set roughly twice the value of β. t_0 and t_d, respectively, indicate the starting time and desired detection time of the attack. (t_d-t_0) is usually set to a small value (e.g., 1s) for quickly detecting an anomaly.

	Number of Failed Sessions, S_n						Total Number of Sessions, T_n						Fraction of Failed Sessions, $F_n = S_n/T_n$							$G_n = F_n - \beta$						
Profiling interval size, Δ_n	Δ_n=5s			Δ_n=10s			Δ_n=5s			Δ_n=10s			Δ_n=5s			m	Δ_n=10s			m	Δ_n=5s			Δ_n=10s		
Protocol	n	time(s)	S_n	n	t(s)	S_n	n	t(s)	T_n	n	t(s)	T_n	n	t(s)	F_n		n	t(s)	F_n		n	t(s)	G_n	n	t(s)	G_n
OpenSSH	1	1-5	…	1	1-10	…	1	1-5	…	1	1-10	…	1	1-5	…	*	1	1-10	…	*	1	1-5	…	1	1-10	…
	2	6-10	…				2	6-10	…				2	6-10	…						2	6-10	…			
	3	11-15	…	2	11-20	…	3	11-15	…	2	11-20	…	3	11-15	…		2	11-20	…		3	11-15	…	2	11-20	…
	4	16-20	…				4	16-20	…				4	16-20	…						4	16-20	…			
	…	…	…	..	…	…	…	…	…	..	…	…	…	…	…		..	…	…		…	…	…	..	…	…
OpenSSL/ TLS	…	…	…	..	…	…	…	…	…	..	…	…	…	…	…		..	…	…		…	…	…	..	…	…

*m = ($\sum F_n$) / (number of observations during n)

Fig. 2. A part of the database format created at a Monitoring stub for profiling.



(a) F_n & m vs. n

(b) G_n & mean of G_n vs. n

(c) Computing the threshold, θ.

(d) Y_n vs. time: Attack is detected when Y_n exceeds θ.

Fig. 3. Computing non-parametric Cusum.

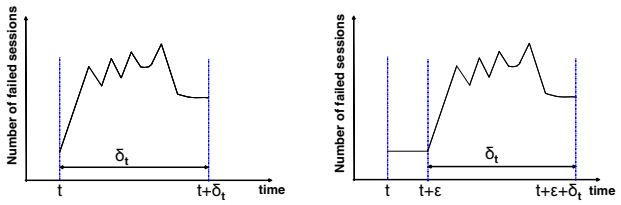
At the detection phase, the MS computes Y_n over time. The Y_n will remain close to zero, i.e., along the horizontal time axis as long as normal conditions prevail in the network. The adopted Cusum algorithm has a computational complexity of $\Theta(1)$ and yields a low memory use [13].

Alert Phase

When an anomaly occurs, the Y_n sequence starts to increase. When Y_n goes beyond θ, an alert is generated by the MS (Fig-3(d)) to the server and the neighboring monitor stubs in the topology. The MS can also slow down the protocol response; to thwart the remote timing attack for instance. Then the MS starts tracing back to identify the attacker which will be described in the next subsection. When the attack stops and the network settles back to the normal stage, Y_n is still likely to have a much bigger value than θ. This will be treated as a false alarm. So a slight modification to the non-parametric Cusum algorithm is required. If the attack stops, we can take note of the downhill curve over some period of time. However a clever attacker may actually lower the attack rate or aggressiveness to evade detection. To counter such a scenario, we employ a cluster based approach. In this approach, as soon as an anomaly is detected, the MS creates a new profile excluding the suspicious clients. That is, we are executing Cusum consecutively (i.e., two Cusum instances). The MS continues to monitor the Cusum distribution based on the initial profiling to see if the attack actually subsides. At the same time, the second monitoring process continues to run using the most recently created profile. Upon detection of an anomaly in this second instance of the Cusum distribution, the MS will build yet another profile. In this way, we dynamically construct new profiles and accordingly adjust the threshold.

Traceback Phase

The traditional traceback schemes include packet marking techniques [14] such as Probabilistic Packet Marking (PPM), ITrace, Deterministic Packet Marking (DPM), and logging techniques such as Source Path Isolation Engine (SPIE). Most of these schemes require the IP-header information. This requirement poses difficulty in tracing back an attacker which sends encrypted packets. In order to use these traditional trace-back methods at the MS level, the MSs may need to decrypt the headers of the attack packets. However, the encrypted packets are to be decrypted only by the target server, not



(a) Number of failed sessions over time seen at MS2.

(b) Number of failed sessions over time seen at MS1.

Fig. 4. No. of failed sessions seen at MS-2 & MS-1.

by the MSs which are only meant to sniff the packets and use previously mentioned schemes to detect attacks. Secondly, decrypting packets at MSs will incur overheads. Therefore, when cryptographic protocols are in use, the commonly known traceback approaches may not provide efficient traceback.

So we adopt an approach similar to that in [15] for tracing back attackers. In this approach, although it may not be possible to track back the actual attacker based on IP due to encrypted headers, we can trace back to the network or domain from which the attacker launched the attack. From thereon, the authorities of that network can hunt down the attack host, which is beyond the scope of our work. Using this approach, in case of cryptographic protocols like SSH or SSL, the MSs monitor failed sessions over time and correlate these patterns, to identify the path of attack traffic. For instance, Fig-1 shows an example where the attacker launches an attack against the cryptographic server. There are 4 MSs distributed over this network. The attack packets traverse the routers which are monitored by MS-1, MS-2, and MS-3. MS-1, which is closest to the victim server, detects the attack, generates alerts and exchanges information with MS-2, its neighboring monitoring stub. MS-2, in turn, sends inquiries to its neighboring MSs, MS-1 and Ms-3. The observation at MS-2 at time t is shown in Fig-4(a). After (t+ξ) seconds, similar record of failed sessions over the same length of time, δ_t, is noticed at MS-1 (Fig-4(b)), where ξ is the propagation delay between MS-1 and MS-2. Thus, we can reconstruct the path back to the attacker's network. Owing to space limitation, we have not included the traceback results but will focus on evaluating the proposed detection scheme.

IV. Performance Evaluation

A. Simulation Setup

A custom simulator was designed with a SSH server and a SSH traffic generator at the client end. The server ran on OpenSuSE Linux 10.1 with 3.2 GHz processor speed and 1 GB memory. The server supported protocol versions were SSH-1 and SSH-2. OpenSSH Traffic Generator/Simulator ran on Windows XP, and the protocol version used was SSH-2.

As shown in Fig-5, M is the number of valid users in the SSH server while N is the number of users in the database including some valid and non-valid users in the simulator. Let (μ = (N-M)/N) denote the "attack aggressiveness". Some valid users may make typing mistakes, and so we consider $\mu \leq 0.05$ to be in the acceptable range.

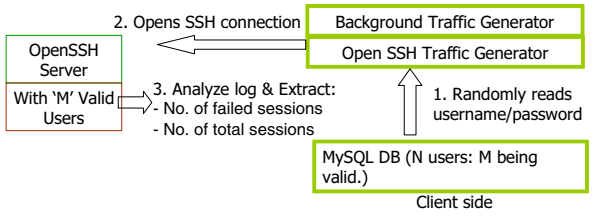


Fig. 5. Simulation setup.

B. Simulation Scenario and Results

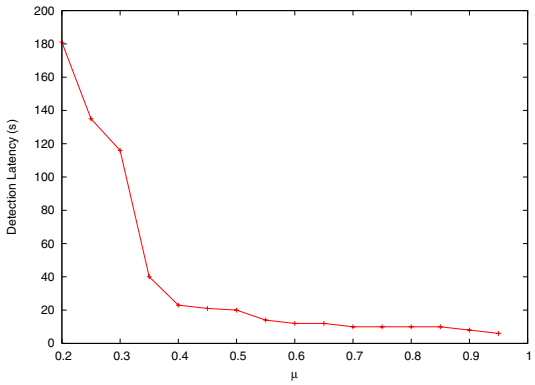
A normal profile was created by using μ=0.05 with sampling interval, Δ=5s. As described earlier, β is selected to be the average of the upper values of F_n. The mean of F_n, m, was 0.036, derived during the profiling period. Threshold θ was calculated using Equation(2). The average β and θ were initially computed to be, respectively 0.31 and 0.34. For lower β and corresponding θ, the time to the first false alarm was rather short, as shown in Table-I. For higher β values starting from 0.30 to 0.32, false alarms were not encountered during the observation period of one hour, thus justifying our selection of β.

TABLE I

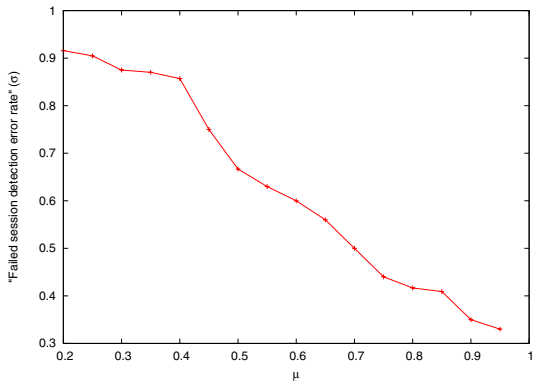
Time to encounter first False alarm (T_F) for different values of β (observation time: 1 hour).

β	T_F(s)		β	T_F(s)
0.05	10		0.22	380
0.08	55		0.25	454
0.1	60		0.28	1,667
0.13	66		0.30	-
0.16	120		0.31	-
0.19	135		0.32	-

To investigate the detection sensitivity of the adopted approach, a number of SSH password attacks were then launched by varying μ from 0.2 to 0.95. For different values of μ, the latency to detect the attack was computed. Fig-6(a) shows the detection delays for detecting attacks with different values of μ using the proposed method. Five simulation runs per μ were performed, and the average values are plotted. The attacks with lower μ took more time to be detected. The detection delays for μ = 0.2, 0.25, and 0.3 were, respectively, 180s, 135s and 116s. The attack profiles with lower μ do not deviate much from the normal profile. This accounts for the high latency for detecting attacks with lower values of μ. For μ-values between 0.3 and 0.4, the latency started to decrease substantially. Between μ = 0.4 and 0.5, the detection delay was approximately 20s on average. The latency started to gradually drop further from μ = 0.55, and was about 15s upto μ = 0.65. The average latency for μ between 0.7 and 0.85 was about 10 seconds. It took the least time, 8s and 6s, respectively, to detect the most aggressive attacks with μ = 0.9 and 0.95. These results indicate that attacks with aggressiveness below 0.4 are harder to detect while those with μ over 0.5 are more easily and quickly detectable.



(a) Latency in detecting attacks of various values of μ.



(b) "Failed session detection error rate" (σ) for various values of μ.

Fig. 6. Simulation results.

Fig-6(b) shows the "Failed session detection error rate", σ, which is expressed by:

$$\sigma = \frac{Number \quad of \quad undetected \quad failed \quad sessions}{Total \quad Number \quad of \quad failed \quad sessions} \ . \qquad (3)$$

As seen from the results, σ was significantly high for the attacks with lower μ. The reason behind this is the overwhelmingly high number of undetected failed sessions accumulated during the high detection-latency in contrast with the moderate number of detected failed sessions during the last sampling interval. For example, the values of σ were 0.91, 0.90 and 0.87, respectively, for μ = 0.2, 0.25 and 0.3. The values of σ decreased gradually along with the aggressiveness of the attack. For attacks with μ = 0.45 and above, the values of σ dropped substantially. The lowest σ (0.33) was encountered during the attack with μ = 0.95.

In summary, these results show that the proposed scheme exhibits reasonably small detection delays. In case of attacks with significantly high attack-aggressiveness, the detection delays were only about two sampling intervals. The "Failed session detection error rate" increases when the system is detecting an attack with low attack-aggressiveness. Apart from this limitation, the proposed scheme achieves effective detection.

V. Concluding Remarks

In this paper we have proposed an Intrusion Detection System to detect attacks against cryptographic protocols. The proposed IDS is based on distributed monitoring stubs which use non-parametric Cusum algorithm to learn normal network scenario and detect anomalies online. We have performed simulations and demonstrated the effectiveness of the detection technique. Our investigations have considered the attack detection delay and the "failed session detection error rate". Furthermore, monitoring stubs are also equipped with traffic-pattern based traceback to track down the attacker. Our future work will extend the proposed scheme to combat against attacks on encrypted protocols in the wireless network environment.

References

[1] D. Brumley and D. Boneh, "Remote Timing Attacks are Practical," in *Proc. of the* 12^{th} *USENIX Security Symposium*, Washington DC, USA, Aug. 2003.

[2] S. P. Joglekar and S. R. Tate, "ProtoMon: Embedded Monitors for Cryptographic Protocol Intrusion Detection and Prevention," *J. Universal Computer Science (JUCS)*, Vol. 11, No. 1, Jan. 2005, pp. 83-103.

[3] V. A. Siris and F. Papagalou, "Application of Anomaly Detection Algorithms for Detecting SYN Flooding Attacks," in *Proc. of IEEE Globecom 2004 (Security and Network Management Symposium)*, Dallas, USA, Nov. 2004.

[4] B. Canvel, A. Hiltgen, S. Vaudenay, and M. Vuagnoux, "Password Interception in a SSL/TLS Channel," in *Proc. Crypto 2003, Advances in Cryptology, LNCS 2729*, Santa Barbara, California, USA, Feb. 2003.

[5] D. Wagner and B. Schneier, "Analysis of the SSL 3.0 Protocol," in *Proc. of Second USENIX Workshop on Electronic Commerce*, Oakland, California, USA, Nov. 1996.

[6] F. Perriot and P. Szor, "An Analysis of the Slapper Worm Exploit," *Symantec Security Response, Symantec White Paper*, Apr. 2003.

[7] R. Oppliger, R. Hauser, and D. Basin, "SSL/TLS session-aware user authentication - OR how to effectively thwart the man-in-the-middle," *Computer Communications*, Vol. 29, No. 12, Aug. 2006, pp. 2238-2246.

[8] A. Stavrou, D. L. Cook, W. G. Morein, A. D. Keromytis, V. Misra, and D. Rubenstein,"WebSOS: an Overlay-based System for Protecting Web Servers from Denial of Service Attacks," *Computer Networks*, Vol. 48, No. 5, Aug. 2005, pp. 781-807.

[9] Available at CERT, "OpenSSL servers contain a buffer overflow during the SSL2 handshake process," *CERT Vulnerability Note #102795*, Jul. 2002.

[10] Available at CERT, "OpenSSL servers contain a remotely exploitable buffer overflow vulnerability during the SSL3 handshake process," *CERT Vulnerability Note #561275*, Jul. 2002.

[11] T. Peng, C. Leckie, and K. Ramamohanarao, "Proactively Detecting Distributed Denial of Service Attacks using Source IP Address Monitoring," in *Proc. of the Third International IFIP-TC6 Networking Conference*, Athens, Greece, May 2004.

[12] H. Wang, D. Zhang, and K. G. Shin, "Change-Point Monitoring for Detection of DoS Attacks," *IEEE Transactions on Dependable and Secure Computing*, Vol. 1, No. 4, Oct. 2004, pp. 193-208.

[13] G. Carl, G. Kesidis, R. Brooks, and S. Rai, "Denial-of-Service Attack-Detection Techniques," *IEEE Internet Computing*, Vol. 10, No. 1, Jan. 2006, pp. 82-89.

[14] Z. Gao and N. Ansari, "Tracing Cyber Attacks from the Practical Perspective," *IEEE Communications Magazine*, Vol. 43, No. 5, May 2005, pp. 123-131.

[15] G. Mansfield, K. Ohta, Y. Takei, N. Kato, and Y. Nemoto, "Towards trapping wily intruders in the large," *Computer Networks*, Vol. 34, No. 4, Oct. 2000, pp. 659-670.